TM-5243/004/00

INTERACTIVE SYSTEMS RESEARCH: FINAL REPORT TO THE DIRECTOR, ADVANCED RESEARCH PROJECTS AGENCY, FOR THE PERIOD 16 SEPTEMBER 1974 to 15 SEPTEMBER 1975

15 NOVEMBER 1975

THIS REPORT WAS PRODUCED BY SDC IN PERFORMANCE OF CONTRACT DAHC 15-73-C-0080, ARPA ORDER NO. 2254, PROGRAM CODE NUMBER 5D30.

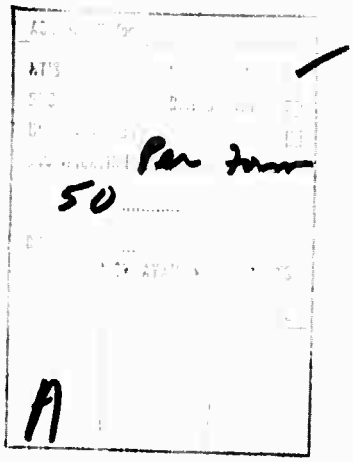
System Development Corporation
2500 Colorado Avenue ▪ Santa Monica, California 90406

TM-5243/004/00

INTERACTIVE SYSTEMS RESEARCH: FINAL REPORT TO THE DIRECTOR, ADVANCED RESEARCH PROJECTS AGENCY, FOR THE PERIOD 16 SEPTEMBER 1974 to 15 SEPTEMBER 1975

15 NOVEMBER 1975

M. L BERNSTEIN
(213) 829-7511, EXT. 6117



THIS REPORT WAS PRODUCED BY SDC IN PERFORMANCE OF CONTRACT DAHC 15-73-C-0080, ARPA ORDER NO. 2254, PROGRAM CODE NUMBER 5D30.

System Development Corporation
2500 Colorado Avenue • Santa Monica, California 90406

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (9) Technical rept. 16 Sep 74 - 15 Sep 75	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) Interactive Systems Research: Final Report to the Director, Advanced Research Projects Agency for the Period 16 September 1974 to 15 September 1975	5. TYPE OF REPORT & PERIOD COVERED: Technical 9/16/74 - 9/15/75
	6. PERFORMING ORG. REPORT NUMBER: (14) SDC TM-5243/004/00
7. AUTHOR(s) (10) M. I. Bernstein	8. CONTRACT OR GRANT NUMBER(s): (15) DAHC15-73-C-0080 ARPA Order-2254
9. PERFORMING ORGANIZATION NAME AND ADDRESS: System Development Corporation, 2500 Colorado Avenue, Santa Monica, California 90406	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: ARPA Order No. 2254, Program Code No. 5D30
11. CONTROLLING OFFICE NAME AND ADDRESS: Information Processing Techniques Office, Defense Advanced Research Projects Agency, Arlinton, Virginia 22209	12. REPORT DATE: (11) 15 November 1975
	13. NUMBER OF PAGES: iii, 72 (12) 78 p.
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Cleared for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

speech-understanding research
natural-language understanding
data-base conversion
lexical semantics
acoustic phonetics

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Progress in developing a prototype computer system for understanding human speech is described. The system is designed to respond to queries regarding characteristics of several hundred naval vessels. This report describes the acoustic-phonetic and lexical-mapping processes of the system in some detail. Also described are two other projects--one to develop an archive of lexical-semantic information on English words and one to develop a system for efficient conversion of data bases from one data management system to another.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

339 900

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

LIST OF FIGURES

(Page iv blank)

TABLE OF CONTENTS (Cont'd)

LIST OF TABLES

1. INTRODUCTION AND SUMMARY

This report to the Advanced Research Projects Agency (ARPA) summarizes System Development Corporation's progress during 1974-1975 in an ongoing program of Interactive Systems Research. During this period, the program included three projects: (1) Speech Understanding Research, (2) Lexical Data Archive, and (3) Common Information Structures. The overall intent of the research is to develop technologies for improved man-machine interaction and for new data management capabilities. Although this report covers the entire year, it emphasizes progress made during the six months from March to September, 1975; an Interim Report (SDC TM-5243/003/00, 15 May 1975) described major accomplishments during the prior six months.

Speech Understanding Research

Our work in speech understanding research is directed toward the construction, by the end of 1976, of a prototype Speech Understanding System. Such a system requires several sources of knowledge about language and its use for particular tasks. These include the parameters of speech sounds, acoustic-phonetic data, and stored information about an ongoing dialogue. SDC is developing the system in cooperation with Stanford Research Institute (SRI), which is primarily responsible for the system's parser and higher-level linguistic processes. SDC is primarily responsible for the overall system implementation and testing and for the modules that process a speaker's input utterances.

The design of our acoustic-phonetic processor reflects the fact that the speech signal is never wholly unambiguous; any attempt to precisely label phones and their boundaries must recognize and allow for considerable ambiguity in mapping the extremely large number of speech sounds into a relatively small set of acoustic-phonetic transcription symbols. Accordingly, in this processor, each acoustic-phonetic frame has multiple labels, and each label is assigned a score. Scores are based on a measure function that is, in turn, based on feature parameters previously developed for each speaker (use).

Late in 1975, a milestone version of the system will be demonstrated. The capabilities of the Milestone System were described in some detail in the Interim Report; this report adds to that description detailed discussions of the now advanced versions of the processes through which the system passes a speaker's utterance in an attempt to derive an acoustic-phonetic representation of the utterance and to map that representation to words in the system's lexicon. Also described is the new programming language and system, CRISP, that we are developing to provide greater capabilities for implementing large portions of the prototype system.

Lexical Data Archive

The Lexical Data Archive Project was begun in 1973 to create a centrally organized archive of lexical semantic information on words in the lexicons being used by ARPA SUR contractors. Files with a considerable amount of data on these words were constructed and are now stored on magnetic tape. The project was terminated in June.

Common Information Structures

The Common Information Structures Project, which was suspended at the end of this year, has implemented a system of languages and translation interfaces for semiautomatic conversion of large data bases from one data-management-system environment into another with minimum cost, effort, and disruption to users. The system was developed over a period of three years and has been successfully tested and demonstrated. Its major advantage over previous designs is that it uses the existing query and generate functions of data management systems as part of the conversion process. This frees the user from having to become familiar with the storage-level and hardware-level data structures of the computer systems involved in the conversion, allowing him to focus on specifying the logical structure of his data base and the types of logical conversions that are necessary to move it from one system to another.

2. SPEECH UNDERSTANDING RESEARCH

2.1 INTRODUCTION

Late in 1974, the first fully integrated prototype of a speech understanding system developed jointly by SDC and the Stanford Research Institute (SRI) was implemented. The task domain of that system was data management on attributes of submarines. The system operated on SDC's Raytheon 704 and IBM 370/145 computers. The Raytheon computer was used to perform an acoustic-phonetic analysis of a digitized speech waveform. The results of this analysis were put into an array of acoustic-phonetic data that we refer to as the A-matrix. The data in the A-matrix are used by lexical mapping procedures to verify the existence of words hypothesized by a "best-first" parser that draws on a set of language-definition (syntax) rules and on components containing semantic and pragmatic (discourse-context) sources of knowledge.

The acoustic-phonetic processing and lexical mapping procedures were essentially the same as those used by SDC in its Voiced-controlled Data Management System, modified to handle a vocabulary of 300 words. The system had a word-string mapping procedure that handles coarticulation between pairs of words.

The goal for this contract year was the Milestone System. The task domain for the Milestone System is data management with an expanded data base containing attributes of submarines, aircraft carriers, and ocean escorts of the US, USSR, and UK. The vocabulary has been extended to 600 words; the system accommodates six speakers, both male and female. Acoustic analysis is performed on PDP-11/40 and SPS-41 computers, interfaced to the IBM 370/145. Refined and augmented acoustic-phonetic analysis includes improved formant tracking, pitch tracking, and vowel-sonorant analysis. Techniques have been developed for handling voiced fricatives and plosives, and improvements have been made to the present programs for handling unvoiced fricatives and plosives. A new programming system, CRISP, will provide efficient arithmetic and array processing, in addition to efficient symbol and list processing, and will substantially increase the address space. Two new mapping procedures have been added: one to handle prosodic features and one to provide word spotting and do lexical subsetting on the basis of robust acoustic cues. Modifications to the syntax include the addition of time and place the conjunction and negation of noun phrases, the use of prosodic attributes and factors (earlier written into the rules), and an allowance for incomplete sentences to function as utterances. Semantic information guides retrieval and prediction in addition to interpretation. The discourse model has been augmented to handle longer dialogue sequences on the basis of protocols gathered in more carefully controlled experiments. System exercising is being guided by formal test and validation procedures that assess each component's contribution to system performance.

2.2 MAJOR ACCOMPLISHMENTS FOR 1974-1975

During the 1974-75 contract year, the SDC SUR group accomplished several major objectives in the development of speech processing algorithms, a comprehensive naval ships data base, and the conduct and analysis of several protocol experiments designed to study the language behavior of naval officers in simulated command situations. Extensive expansions of a ships data base were coordinated with technical and military personnel of the Naval Electronics Laboratory Center (NELC), San Diego. Several important algorithms were developed for processing speech waveforms, including fundamental frequency extraction, formant frequency analysis, segmentation and labeling, and various procedures for word and phrase pattern-matching. A new programming language and system, CRISP, is being developed to provide more powerful capabilities than are provided by LISP and its derivatives.

2.2.1 Language Behavior in Naval Operations

The conduct of protocol experiments represents an important aspect of our system-building strategy. The dialogues obtained from these experiments are the basis for our decisions regarding:

1. discourse context,
2. syntax,
3. vocabulary,
4. lexical selection of phonetic base forms,
5. prosodics, and
6. data base content.

Several protocols were gathered at the Naval Postgraduate School in Monterey, California, during July, 1974. These were followed by a further protocol experiment in the SDC SUR laboratory. The design of a new set of experiments was then worked out with technical and military personnel at the Naval Electronics Laboratory Center (NELC) in San Diego, California. The experiments were conducted with military personnel at NELC in May, 1975. The subjects were high-ranking naval officers with extensive experience in command operations. Each subject was given the problem of assessing the potential strength and combat readiness of ships in the U.S. Sixth Fleet during a simulated crisis situation in the eastern Mediterranean. Updated "intelligence" reports concerning the movements of foreign ships in the Mediterranean and adjacent areas were issued to each subject at 10-15 minute intervals during the conduct of each hour-long experiment.

Orthographic transcriptions of the recorded protocol dialogues have been used to identify necessary syntax, vocabulary, and data base extensions to the system, and have provided useful information about discourse context. The transcriptions

also served as prompting material for subjects who participated in an experiment conducted in the SDC laboratory. The results from the latter experiment were transcribed orthographically at SDC and phonetically at SCRL. These phonetic transcriptions guided the selection of lexical base forms and their accompanying phonology. Also, the phonetic transcriptions, along with acoustic analyses of the utterances in the dialogues, will be useful in the analysis of prosodic phenomena.

2.2.1.1 Computer Processing of Protocol Recordings

The major results of the computer processing of the protocol recordings are KWIC concordances, type counts, and "word"-lists sorted by frequency. Concordances were found very useful in the analysis of our current protocols. SDC currently has the following capabilities for concordance generation:

1. KWIC index for orthographic text (Figure 2-1).
2. KWIC index for phonemically transcribed text (Figure 2-2).
3. KWIC index for individual phonemes (Figure 2-3).
4. A concordance in which keywords are displayed together with the entire sentence in which they appear (Figure 2-4).

All versions provide basic statistics of the text processed, e.g., number of sentences in the text, total number of tokens (words or phonemes, as the case may be), number of types, type/token ratio, frequencies, percentage frequencies, etc.

The four protocol experiments yielded nine recordings of natural speech for nine different speakers, comprising a total of 955 utterances, a total of 9461 orthographic tokens, 1791 orthographic types, with an overall type/token ratio of approximately 19%. It should be noted that the term "utterance" is used somewhat loosely; it covers single-word fragments such as "all right" and "O.K.," sentence fragments, complete sentences, and, in some cases, whole paragraphs. This variety is due to the fact that, during the first two experiments, the subjects' queries normally took the form of single sentences, while in the last two experiments, especially the experiment involving NELC personnel, the dialog was much more complex--a natural consequence of the increased complexity of the scenario and the data base. It was found convenient for processing purposes to consider the whole query as a unit, rather than break it up into individual sentences. As far as the concordances are concerned, an utterance refers to what was said by the subject between responses from the system.

The orthographic KWIC index (Figure 2-1) for our current protocol files has highlighted frequently recurring sentence types and other grammatical constructions. For example, out of a total of 220 sentences, 62--i.e., almost a third--begin with "How many <NP>..." and another third begin with "What is <NP>..." and "What's <NP>..." By taking data such as these into account, the parser can focus on the more likely paths first.

```
                    NUMBER OF REACTORS?                                  003014
                WHICH CLASS OF SUBMARINE HAS THE GREATEST NUMBER OF MISSILE LAUNCHERS?  003038
              GIVE ME THAT LIST OF SUBMARINES AGAIN.                     003044
          WHAT IS THE SUBMERGED SPEED OF THE ALBACORE?                   003048
    WHAT IS THE SURFACE DISPLACEMENT OF THE ETHAN ALLEN?                 003005
          WHAT IS THE THE LENGTH OF THE ETHAN ALLEN?                     003011
              THE SUBMERGED SPEED OF THE ETHAN ALLEN?                    003049
              THE SUBMERGED SPEED OF THE ETHAN ALLEN?                    003059
                                OF THE GEORGE WASHINGTON?                003006
                    AND THE LENGTH OF THE GEORGE WASHINGTON?             003012
          THE SURMERGED SPEED OF THE GEORGE WASHINGTON?                  003060
OF MISSILE LAUNCHERS AND QUANTITY OF THE LAFAYETTE.                      003002
                  WHAT IS THE LENGTH OF THE LAFAYETTE?                   003004
          WHAT IS THE SUBMERGED SPEED OF THE LAFAYETTE?                  003047
          WHAT IS THE SUBMERGED SPEED OF THE LAFAYETTE?                  003058
                                OF THOSE, WHAT IS THE MAXIMUM AND MINIMUM COMPLEMENT?  003068
```

Figure 2-1. Partial Orthographic KWIC Index for "of" in Various Contexts

```
OF
      A AH:2DX IH:0Z DHAX:0 SAX:0HMER:2JH SPIY:2D AX:0 DHIH:0 AE:2LBAX:0KOW:1R                    003048
                     DHAX:0 SAX:0HMER:2JH SPIY:2D AX:0 DHIH:0 IY:2THEN:0 AE:2L(IX/IH):0N         003059
                     DHAX:0 SAX:0BMER:2JH SPIY:2D AX:0 DHAX:0 JOW:2RJH W(AO/AA):2SHIH:0NT(IX/IH):1N 003060
:0 SAX:0HMER:2JH SPIY:2D AE:1N DHAX:0 NAH:2MHUR:0 AX:0F TOW:1RPIY:2DOW:1 TUW:2HZ PUR:0 DHAX:0 JOH:2R 003057
          LIH:2ST AO:2L KLAE:2S(IX/IH):0Z AX:1V AX:0V NUW:2KL(IY/IH):0UR:0 BEL:0IH:2STIH:1K MIH:2S 003001
 DIH:1SPLEY:2SM(IX/IH):0NT LEH:2NXKTH NAH:2MBUR:0 AX:0V TOW:1RPIY:2DOW:1 TUW:2HZ NAH:2MBUR:0 AX:0V R 002
MBUR:0 AX:0V TOW:1RPIY:2DOW:1 TUW:2HZ NAH:2MBUR:0 AX:0V RIY:1AE:2KTUR:0Z NAH:2MBUR:0 AX:0V MIH:2SEL: 002
BZ NAH:2MBUR:0 AX:0V RIY:1AE:2KTUR:0Z NAH:2MBUR:0 AX:0V MIH:2SEL:0 L(AO/AA):2NCHUR:0Z  EN:0 KW(AO/AA 003002
:0 L(AO/AA):2NCHUR:0Z  EN:0 KW(AO/AA):2NAX:0TIY:1 AX:0V DHAX:0 L(AO/AA):1FEH:0YEH:2T              003002
          WAH:2DX  (IX/IH):0Z DHIY:2 LEH:2NXKTH AX:0V DHAX:0 LAA:1F(IH/EH):0YEH:2T                 003004
DHAX:0 SER:2F(IH/IX):1S DIH:1SPLEY:2SM(IX/IH):0NT AX:0V DHIH:0 IY:2THEN:0 AE:2L(IX/IH):0N         003005
                                                  AX:0V DHAX:0 JHOW:2RJH WAA:2SHIH:0NT(IX/IH):1N 003006
      WAH:20X IH:0Z DHAX:1Q DHAX:0 LEH:2NXKTH AX:0V DHIH:0 IY:2THEN:0 AE:2L(IX/IH):0N             003011
                    AE:2N DHAX:0 LEH:2NXKTH AX:0V DHAX:0 JHOW:2RJH WAA:2SHIH:0NT(IX/IH):1N       003012
                              NAH:2MHUR:0 AX:0V RIY:1AE:2KTUR:0Z                                 003014
                              NAH:2MBUR:0 AX:0V MIH:2SEL:0 L(AO/AA):2NCHUR:0Z                    003015
1N HHAE:1Z DHIH:0 GREY:2DX(IX/IH):0ST NAH:2MHUR:0 AX:0V MIH:2SEL:0 L(AO/AA):2NCHUR:0Z             003037
                          WHIH:2CH KLAE:2S AX:0V SAH:2HMUR:0IY:1N HHAE:1Z DHAX:0 GRE:2TIH:1S 003038
R:0IY:1N HHAE:1Z DHAX:0 GREY:2TIH:1ST NAH:2MHUR:0 AX:0V MIH:2SEL:0 L(AO/AA):2NCHUR:0Z              003038
N HHAE:1Z  DHAX:0 GREY:2DX(IX/IH):0ST NAH:2MBUR:0 AX:0V MIH:2SEL:0 L(AO/AA):2NCHUR:0Z              003040
                    GIH:2V MIY:1 DHAE:1T LIH:2ST AX:0V SAH:2BMUR:0IY:1NZ  :0GEH:2N              003044
1N HHAE:2Z DHAX:0 GREY:2DX(IX,IH):0ST NAH:2MHUR:0 AX:0V TOW:1RPIY:2DOW:1 TUW:2VZ                 003046
      WHAH:2DX IH:0Z DHIY:1 SAX:0HMER:2JH SPIY:2D AX:0V DHAX:0 L(AO/AA):1FEH:0YEH:2T             003047
                     DHAX:0 SAX:0BMER:2JH SPIY:2D AX:0 DHIH:0 IY:2THEN:0 AE:2L(IX/IH):0N         003049
        AH:1 GIH:2V MIY:2 DHIY:2 AH:1 NAE:2MHUR:0 AX:0V MIH:2SEL:0 L(AO/AA):2NCHUR:0Z DHAX:0 SAX:0BM 003057
      WHAH:2DX IH:0Z DHAX:0 SAX:0BMER:2JH SPIY:2D AX:0V DHAX:0 L(AO/AA):1FEH:0YEH:2T               003058
                                                  AX:0V DHOW:2Z WAH:2T IH:1Z DHAX:0 NAE:2KSMAX:1M AE 003068
SAH:2BMUR:0IY:1N HHAE:2Z AX:0 KAA:2MPLAX:0NIX:1NT AX:0V AX:0 HHAH:2NDXUR:0D EN:0 FAY:2V             003069
                    LIH:2ST AO:2L KLAE:2S(IX/IH):0Z AX:1V AX:0V NUW:2KL(IY/IH):0UR:0 BEL:0IH:2STIH:1K 003001
```

Figure 2-2. Partial Phonemically Transcribed KWIC Index for "of" in Various Contexts

```
                                  # DH AX:O # IY:2 IH EN:O # AE:2 L IX:O N #                     3  43
M ER:2 JH # S P IY:2 D # AX:O V # DH IH:O # IY:2 IH EN:O # AE:2 L IX:O N #                       3  49
2 D OW:1 # T UW:2 B Z # D AX:O Z # DH AX:O # IY:2 IH EN:O # AE:2 L IX:O N # HH AE:2 V #          3  54
B M ER:2 JH # S P IY:2 D # AX:O # DH IH:O # IY:2 IH EN:O # AE:2 L IX:O N #                       3  59
          # AE:O N # HH AW:2 # M EH:2 N IY:O # IY:2 IH EN:O # AE:2 L IX:O N Z # AA:1 R # DH UR:1 # 3  16
1 TH # Q AX:O # B IY:2 M # L EH:2 S # DH IX:1 N # IH ER:2 DX IY:O # F IY:2 T #                   3  62
AE:2 V # L EH:2 NX K TH # L EH:2 S # DH IX:O N # IH ER:2 DX IY:O # F IY:2 T #                    3  63
1 # W IH:1 TH # B IY:2 M # L EH:2 S # DH IX:O N # IH ER:2 DX IY:O # F IY:2 T #                   3  64
Z # W IH:1 TH # B IY:2 M # L EH:2 S # DH IX:C N # IH ER:2 DX IY:O # F IY:2 T #                   3  65
T UW:2 B Z # D AX:O Z # DH AX:O # G AH:2 P IY:O # IH R IY:2 # HH AE:2 V #                        3  53
```

Figure 2-3. Partial KWIC Index for Individual Phc.ıeme "TH"

```
HUNDRED
                3  69   WHICH WHICH SUBMARINE HAS A COMPLEMENT OF A HUNDRED AND FIVE?
                FREQUENCY=     1

I
                1  64    AM I SUPPOSED TO TELL YOU WHAT I CHOSE?
                1  64   AM I SUPPOSED TO TELL YOU WHAT I CHOSE?
                1  66   I CHOSE GEORGE WASHINGTON CLASS SUBMARINE.
                FREQUENCY=     3

I'M
                1  81     I'M SORRY, LESS THAN 30 FEET AND HAS A.
                FREQUENCY=     1

II
                2  43   AND, HOW MANY TORPEDO TUBES AND MISSILE LAUNCHERS FOR THE HOTEL II?
                2  61   WHAT'S THE SPEED OF THE GUPPY III AND GUPPY II, SUBMERGED?
                2  62   WHAT'S THE LENGTH OF THE GUPPY III AND THE GUPPY II?
                FREQUENCY=     3

IIA
                2  58   GUPPY IIA?
                FREQUENCY=     1

IIA'S
                2  59   HOW MANY GUPPY IIA'S DO WE HAVE?
                FREQUENCY=     1

III
                2  57   GUPPY III?
                2  61   WHAT'S THE SPEED OF THE GUPPY III AND GUPPY II, SUBMERGED?
                2  62   WHAT'S THE LENGTH OF THE GUPPY III AND THE GUPPY II?
                FREQUENCY=     3

III'S
                2  60   HOW MANY GUPPY III'S?
                FREQUENCY=     1
```

Figure 2-4. Partial Concordance of Keywords with Sentences

The KWIC index routine for phonemically transcribed text is designed to group together all phonetic variants of the same word under its orthographic representation. For example, under "of" (see Figure 2-1) we found the following variants: AX:O, AX:OF, AX:OV, and AX:1V; under "the" we get: DHAX:O, DHIH:O, DHIY:1, DHIY:2. Such a concordance provides a check on the phonological rules component, whose function is to generate the variants likely to be encountered in a speech situation. It also allows us to select the most commonly used phonetic spelling to be used in the mapper for a first try.

The KWIC index for individual phonemes (in ARPABET transcription) has proved of interest to all those concerned with the acoustic-phonetic processing component of the system. For example, the distribution of vowel contexts for initial plosives, the relative importance of final plosives, and the voicing context of /ð/ were items of immediate interest to the researcher working on fricative and plosive analysis. Furthermore, the table ranking phonemes by frequency of occurrence (see Table 2-1) suggests the most pressing areas of research. In the body of protocol sentences analyzed, the phoneme /n/ appears at the top of the list, closely followed by /ə/, /s/, /i/, /m/, /z/, /d/, /ð/, /e/. A survey of related work revealed that the phoneme frequency distribution in our protocol sentences largely matches the distributions in Denes [1] and Shoup [2]. We therefore feel justified in using the output of this frequency study to guide our acoustic-phonetic research. Particularly, it is important to note that the phones /n/, /m/, /l/, and /r/ have a high frequency of occurrence. They exert a great influence on the vowels in their immediate vicinity. Therefore, research on the coarticulation effects between these phones and neighboring vowels was one of our primary tasks for this year.

A research plan for a joint SDC-SRI study of prosodic features and their use in a speech understanding system was developed. Under this plan, SDC performed acoustic processing on the dialogues obtained from the protocol experiments. The first experiment dealt with word duration. A-matrices were prepared for each utterance of the protocol experiment recorded in September, 1974. There were 69 utterances in the protocol. Word and pause durations were determined on the basis of the A-matrix parameters described above. This information was input to a program that created a new file in which each word and pause occurring in the protocol appeared with its duration. For example, utterance #44 in the protocol, which in the original file reads:

"Give me that list of submarines again"

now reads:

"give17 me08 that18 list28 of07 submarines54 again45"

The numbers following each word refer to the number of 10-msec. segments the word spans. For example, the word "give" in this sentence is 17 segments, or 170 msec., long. The new file thus obtained was put through the KWIC indexing routine that groups all similar words together and displays them in their context (see Figure 2-5).

TABLE 2-1. PROTOCOL ANALYSIS: PHONEME FREQUENCY COUNT

Frequency	%Frequency	Phoneme (ARPABET Representation)	Phoneme (IPA Representation
1	0.04	EM	syl m,m
9	0.40	AA	ɑ
9	0.40	Q	ʔ
10	0.45	WH	ʍ
12	0.54	NX	ɳ
13	0.58	G	g
14	0.63	UH	ʊ
15	0.67	EN	syl n,n
16	0.72	SH	s or ʃ
17	0.76	AY	aI or ay
17	0.76	CH	ʧ
17	0.76	ER	ɝ
22	0.99	F	f
22	0.99	W	w
23	1.03	EY	eI or ey
24	1.08	TH	θ
26	1.17	AW	aU or aw
26	1.17	UW	u
27	1.21	AO	ɔ
27	1.21	EL	syl l,l
28	1.26	DX	flapped t,ɾ
29	1.30	Y	y
31	1.39	JH	ʤ
33	1.48	K	k
33	1.48	P	p
36	1.62	OW	o
49	2.21	R	r
50	2.25	IX	ɨ
52	2.34	EH	ɛ
54	2.43	HH	h
55	2.48	V	v
56	2.52	AH	ʌ
61	2.75	UR	ɚ
66	2.97	AE	æ
66	2.97	L	l
68	3.06	B	b
76	3.43	DH	ð
78	3.52	D	d
81	3.60	Z	z
90	4.06	IH	ɪ
108	4.87	T	t
114	5.14	M	m
122	5.50	IY	i
134	6.04	S	s
145	6.54	AX	ə
153	6.90	N	n

```
                                                                                                                  003046
27 THE10 GREATEST36 NUMBER31 OF02 TORPEDO59 IUBES43                                                               003052
                    HOW15 MANY16 TORPEDO49 IUBES43 DOES26 THE10 LAFAYETTE52 HAVE31                                003002
7 LENGTH31 PAUSE039 NUMBER28 OF08 TORPEDO50 IUBES47 PAUSE014 NUMBER31 OF05 REACTORS64 PAUSE070 NUMBE              003013
           PAUSE004 HOW18 MANY20 TORPEDO62 IUBES59 PAUSE053 AH25 DOES31 THE13 ETHAN29 ALLEN46 AND17               003020
                    FOR03 THE09 SOVIET48 UNION30 PAUSE009                                                         003023
ISSILE31 SUBMARINES65 DOES19 THE10 SOVIET40 UNION44 HAVE28                                                        003025
AINING44 SUBMARINES63 PAUSE038 DOES18 THE09 UNITED26 STATES38 HAVE29                                              003019
AUSE023 MISSILE31 SUBMARINES58 DOES23 THE06 UNITED27 STATES38 HAVE21                                              003022
UCLEAR59 ATTACK40 SUBMARINES56 DOES20 THE05 UNITED27 STATES35 HAVE30                                              003001
19 NUCLEAR45 BALLISTIC56 MISSILE44 PAUSE039 UNITED28 STATES31 SUBMARINES69                                        003028
UIDED36 MISSILE26 SUBMARINES59 DOES22 THE05 UNITED28 STATES33 HAVE35                                              003024
 PAUSE033 FLEET32 SUBMARINES59 DOES31 THE09 UNITED29 KINGDOM37 HAVE29                                             003029
USE006 TRAINING41 SUBMARINES61 DOES18 THE04 UNITED29 STATES30 HAVE27                                             003030
1 MANY18 DIESEL31 SUBMARINES58 DOES18 THE03 UNITED33 STATES32 HAVE27                                             003021
                  AND37 THE06 UNITED34 KINGDOM37 PAUSE004                                                        003063
075 HOW15 MANY41 PAUSE063 SUBMAR39 PAUSE007 UNITED36 STATES17 SUBMARINES52 PAUSE024 HAVE32 PAUSE014              003064
          SUBSET63 ON19 ALL33 PAUSE044 UNITED41 STATES35 SUBMARINES79 PAUSE096 AH23 PAUSE009 W1               003062
        LIST32 ALL19 SUB37 PAUSE022 ALL15 UNITED44 STATES31 SUBMARINES67 PAUSE046 WITH37 PAUSE089              003017
             AND54 HOW18 MANY35 GEORGE30 WASHINGTONS61 PAUSE064                                                   003010
           AND74 THE53 PAUSE019 GEORGE36 WASHINGTON48 PAUSE014                                                    003055
028 TORPEDO44 TUBES29 DOES10 THE08 GEORGE29 WASHINGTON48 HAVE25                                                   003050
      PAUSE013 AH21 PAUSE012 THE14 GEORGE31 WASHINGTON51                                                          003006
           PAUSE005 OF13 THE06 GEORGE32 WASHINGTON52                                                           003013
DOES31 THE13 ETHAN29 ALLEN46 AND17 GEORGE29 WASHINGTON52 HAVE29 RESPECTIVELY79                                    003042
                        THE08 GEORGE27 WASHINGTON53                                                           003057
OF14 TORPEDO51 TUBES36 FOR06 THE08 GEORGE34 WASHINGTON53 PAUSE006                                                 003060
E12 SUBMERGED36 SPEED21 OF02 THE07 GEORGE29 WASHINGTON55 PAUSE057                                                 003012
05 AND38 THE09 LENGTH58 OF20 THE09 GEORGE29 WASHINGTON57 PAUSE024                                                 003008
    FOR16 WHICH49 FOR08 WHICH22 SUBMARINE57 WAS14 THAT32                                                          003009
                                AH60 WHAT09 IS12 THE10 SUBMERGED48 DISPLACEMENT56 FOR08 THE1                      003005
                                     WHAT11 IS18 THE07 SURFACE42 DISPLACEMENT67 OF11 THE04 E                      003068
                   OF05 THOSE28 PAUSE009 WHAT11 IS14 THE09 MAXIMUM71 PAUSE003 AND34 PAUSE020 MIN                  003004
                                         WHAT12 IS18 THE33 PAUSE012 LENGTH33 OF10 THE04 LAFAYETT                  003045
                                         WHAT13 SUBMARINE61 HAS30 THE11 FASTEST50 SUBMERGED57 SP                  003011
                                         WHAT16 IS13 THE36 PAUSE067 THE24 LENGTH44 PAUSE019 OF20                  003048
                                         WHAT19 IS11 THE11 SUBMERGED44 SPEED24 OF02 THE04 ALBACO                  003040
                                         WHAT22 SUBMARINE71 HAS24 THE13 GREATEST39 NUMBER32 OF04                  003058
                                         WHAT24 IS19 THE13 SUBMERGED45 SPEED22 OF07 THE08 LAFAYE
```

Figure 2-5. Partial KWIC Index with Word Durations

Table 2-2 shows the mean of terms that occurred more than 10 times in the corpus of utterances. Durations for the same term often vary as much as twice their shortest duration. The durations of short function words show much larger variations than those of context words or compound terms.

TABLE 2-2. PROTOCOL ANALYSIS: SAMPLE FREQUENCY AND DURATION DATA FOR TERMS OCCURRING MORE THAN 10 TIMES

TERM	FREQUENCY	MINIMUM DURATION (NO. OF 10-MSEC. SEGMENTS)	MAXIMUM DURATION	MEAN
The	64	3	53	11.67
Of	29	2	31	8.93
How many	25	24	57	45.12
Submarines	21	40	91	60.57
Have	21	15	38	26.9
Missile	14	26	44	33.64
Number	11	26	51	33.29
Submerged speed	8	54	102	74.13

We anticipate that duration data of this kind will eventually be used by several components of the system, in particular by the mapper, where they could become the basis for one of its subsetting functions. It is conceivable that information on word duration could also become part of the user model, along with the speaker-dependent vowel tables and other such data.

2.2.1.2 The Naval Ships Data Base

An early version of SDC's Vocal Data Management System (VDMS) contained a data base of information about the submarine fleets of the United States, the Soviet Union, and the United Kingdom. A few of the preliminary protocol experiments were conducted with this data base--specifically, those run at the Naval Postgraduate School and a follow-up experiment conducted at SDC. The simplicity of this data base restricted the variety of questions that could be asked during an experiment; it became obvious that in order to obtain more meaningful dialogues

TABLE 2-3. SAMPLE DATA BASE ENTRIES FOR THE USS CONSTELLATION

FIELD NAME	FIELD CONTENTS
Country	USA
Ship.type	CVA
Hull.#	CVA64
Name	Constellation
Class	Kitty Hawk
Quantity (in class)	4
Readiness (in hours)	Ø
Location (port name, sea)	Sea
Longitude (° for E, -° for W)	15
Latitude (°N)	34
Heading (direction in °)	Unknown
Fuel.Status (% full)	70
Displacement (subm displ. for subs, std. displ. for surface ships) (tons)	60,100
Draft (feet)	35.4
Length (feet)	1072.5
Beam (feet)	129.5
ASW (anti-submarine warfare)	None
AA (anti-aircraft)	2 Twin Terriers
SS (surface to surface)	None
Torpedo tubes	None
Aircraft	Approx. 85
Propulsion (nuclear/conventional)	Conventional
Engines	4 geared turbines, 8 boilers
Max. crusing speed (knots)	35
Complement (total)	2795
Builder	New York Naval Shipyard
Laid.down	Sept. 14, 1957
Launched	Oct. 8, 1960
Commissioned	Oct. 27, 1961

a larger and more realistic data base was needed. This need raised the question of how to extend the data base. Should the primary consideration be usefulness in the study of speech, or should it be utility in the real world? That is, should the data base be extended so that it would give rise to linguistically more interesting dialogues irrespective of its usefulness in some real-world situation, or should it be extended with some practical application in mind? The first alternative retains a "toy" domain but is satisfactory from the point of view of studying speech understanding; the second has the advantage of applicability in the real world, but requires considerable effort not only in building the data base, but also in extending the language-handling capabilities of the system. Since the primary objective of the SUR project is to study speech understanding rather than to build a practical system, the first alternative was favored. The close collaboration of technical and naval experts at NELC was sought, and a great deal of effort was spent in extending the data base to make it more realistic, without, however, aiming at immediate applicability.

The former submarine data base was extended to include a variety of approximately 250 ships, such as aircraft carriers, destroyers, and frigates. Moreover, many more attributes were added for each ship. These now include 32 attributes, including location, fuel status, readiness status, and armament. A sample content entry from this expanded data base is shown in Table 2-3.

2.2.2 Speech Processor Component Development

The SDC-SRI speech system comprises three major processing components:

1. The acoustic-phonetic processor, which extracts acoustic parameters from the speech waveform of an utterance and applies rules to these parameters to generate an acoustic-feature description of the utterance;

2. The lexical mapping procedure, which attempts to match words and phrases hypothesized by the parser with data generated by the acoustic-phonetic processor; and

3. The linguistic processor, developed by SRI, which includes a parser that makes hypotheses about the content of an utterance using syntax rules, semantics, and pragmatics.

The parser hypothesizes words that it considers highly likely to occur in an utterance. The hypothesized words are transmitted to the lexical mapping procedure, which extracts an idealized pronunciation of the word from a lexicon. This idealized pronunciation, along with a set of alternate pronunciations (as generated by phonological rules) is then mapped against the acoustic-feature strings extracted from the utterance by the acoustic-phonetic processor. When a word has been successfully mapped, this information is returned to the parser, which then makes further hypotheses about other words in the utterance. The

process continues until the parser decides that the string of words it has found form a complete utterance. Finally, a mechanism is used to extract the appropriate response from the data base. If the parser is not able to derive a complete understanding, a partial parsing of the utterance is returned to the user. A complete description of the parser's operation is contained in [3]. The remainder of this section describes the operation of the acoustic-phonetic processor and the lexical mapping procedure, the system hardware/software configuration, and the CRISP programming language.

2.2.2.1 Acoustic-Phonetic Processing

The incoming speech signal is a time-varying sound-pressure waveform. A preliminary machine representation of this waveform is obtained by passing it through an analog pre-sampling filter and digitizing the filter output at the rate of 20,000 samples per second. Each of the resulting samples is represented as a 12-bit integer. Thus, for each second of speech, 240,000 bits of data are generated. This form of the data does not explicitly contain any of the important features of the waveform that are needed for subsequent processing. The initial goal, therefore, is to generate a parametric representation of the waveform that will contain a number of useful acoustic features.

A variety of parameters can be extracted from a speech waveform. They include frequency, amplitude, and pitch characteristics. Some relate directly to the speech production process, while others relate to the auditory processes involved in speech perception. We have chosen to parameterize the waveform on the basis of speech-production characteristics--first, because a substantial body of knowledge about vocal-tract resonance characteristics has been accumulated over the past 30 years through the study of sound spectrograms and, second, because recently developed signal-processing techniques have led to the development of accurate and computationally efficient procedures for deriving vocal-tract resonance characteristics.

Parameterization is initiated with the calculation of the root mean square (RMS) value for each 10-msec. frame of speech. This calculation is followed by fundamental frequency extraction, formant frequency analysis, syllable segmentation, phrase segmentation, and other analyses. These analyses are described below.

Fundamental Frequency Extraction

A number of algorithms have been devised for extracting fundamental frequency (F0), or pitch, from digitized speech signals. These algorithms are often used both to estimate pitch and to distinguish between voiced and unvoiced speech. Pitch-tracking algorithms may be divided into two broad classes: frequency domain and time domain.

1. Frequency-domain algorithms extract spectra of some sort, whether Fast Fourier Transform (FFT), Linear Predictive Coding (LPC), cepstrum, or autocorrelation. The spectra are then analyzed to find the value of F0. Frequency-domain pitch trackers tend to be slow because the extraction of spectra is ordinarily a slow process unless special-purpose hardware, or a very fast computer, is available.

2. Time-domain algorithms do not extract spectra. Instead, they attempt to identify glottal pulses in the speech signal and calculate pitch values from the distance between the pulses. Time-domain pitch trackers [4,5,6] are the fastest type, running at real time or less even on minicomputers. Unfortunately, they are too inaccurate for many applications.

The cepstrum [7,8,9] method, which requires the equivalent of two FFTs per time frame, is generally considered to be the most accurate. The characteristics of the voicing source are examined after they are separated from the effects of vocal-tract resonances. The cepstrum is resistant to phase and amplitude distortion of the signal but is sensitive to noise and requires more computation time than the other methods. The normalized error function of the LPC [10,11] can be used to unveil the train of glottal pulses in voiced speech, which can then be tracked as in the time-domain pitch trackers. FFTs [12,13] have been used to extract pitch by analyzing the pattern of peaks in the FFT spectrum. The autocorrelation technique [14,15] generates spectra quantized in period rather than frequency, which is convenient for looking at low-frequency phenomena like pitch. Autocorrelation spectra are computationally simple but do not resolve fundamentals over harmonics and subharmonics as strongly as do more conventional spectra.

The differences between frequency-domain and time-domain pitch trackers are particularly sharp in terms of speed and accuracy. Frequency-domain analysis extracts whatever periodicity information is present, degrading gracefully in the presence of noise or distortion. On the other hand, with the fast time-domain pitch trackers, it is assumed that the glottal pulse is necessarily a prominent feature of the speech signal, or that the wave shape of a pitch period changes only slowly from time to time. Unfortunately, these assumptions do not hold for many phonetic environments, speakers, and recording conditions. The pitch tracker developed by Gillmann [16] is a frequency-domain pitch tracker that approaches the speed of the time-domain pitch trackers without giving up the higher reliability expected of the frequency-domain approach. It operates in three phases:

1. Down-sampling. A digital filter is used to reduce the original speech signal (which has been sampled at the rate of 20,000 samples per second) to 2,000 samples per second, thus removing many frequencies that lie outside the range of possible fundamentals and improving the speed of the program.

2. Autocorrelation and pitch extraction. An autocorrelation spectrum with a window size of 50 msec. is taken every 10 msec. An algorithm examines these spectra and picks peaks from them. The algorithm considers the possibility of octave errors [17] (mistaking a harmonic or subharmonic for the fundamental) and deals with them. To reduce frequency quantization, a parabola is fitted to the peak chosen in the spectrum, and the theoretical peak of this parabola is used as the pitch value.

3. Editing. The F0 values obtained above are passed through a median smoother to eliminate anomalous values, and then a heuristic pitch-track editor attempts to remove any remaining errors. Figure 2-6 illustrates the results of the program applied to the utterance "The U. S. has Lafayettes." Note the discontinuities of the contour occurring during the unvoiced portions of the utterance (/s/, /z/, /f/, /s/).

Each 10-msec. frame is labeled voiced if a pitch value has been assigned to it and labeled unvoiced if a pitch value of zero has been assigned.

Spectral Analysis and Formant Frequency Analysis

Spectral analysis using a Linear Predictive Coding (LPC) algorithm (see, e.g., Markel [18]) is applied to a 25.6-msec. frame of speech centered at each voiced 10-msec. frame. The major advantage of using LPC techniques for spectral analysis stems from the fact that the underlying model from which a spectral approximation is obtained has a z-transform given by

$$A(z) = \frac{1}{1 - \sum_{k=1}^{p} a_k z^{-k}}$$

This all-pole representation provides a realistic approximation to the vocal-tract-resonance characteristics of most voiced speech sounds. The peaks in the spectrum correspond to poles of A(z) and are close approximations to the formant frequencies of voiced speech.

Considerable information is obtainable from formant frequency values taken as a set of individual 10-msec. frames, but a wealth of additional knowledge results from construction of a piecewise-continuous time function called a formant track. This information is critical to the development of acoustic-phonetic algorithms that describe the coarticulation processes involved in changing from one speech sound to another. An extremely complex procedure is required to construct a formant track because of discontinuities in formant

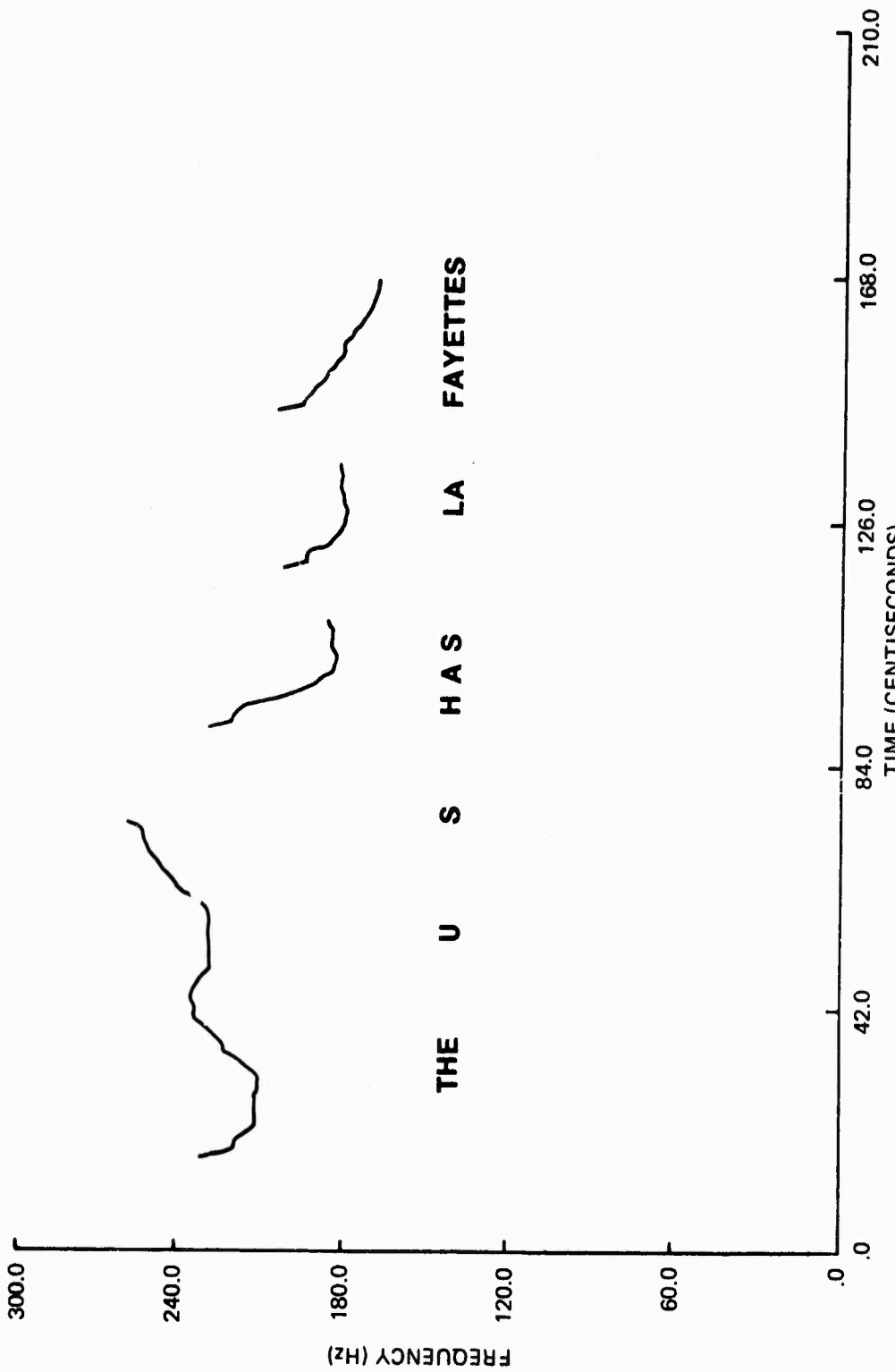


Figure 2-6. Fundamental Frequency Contour of the Utterance "The U.S. has Lafayettes."

structure due to changes from voiced to unvoiced speech and because, even within voiced areas, discontinuities due to the appearance of complex speech sounds (such as nasal murmurs) can occur. A program developed by Kameny, Gillmann, and Brackenridge [19] is used to construct the formant track. The result is similar to a digital representation of a sound spectrogram or "voiceprint" but with the exact frequency information known. The program assigns frequency values to each of the first three formants for each 10-msec. voiced segment of continuous speech. Its input parameters are fundamental frequency, RMS energy, and the frequencies of up to five spectral peaks below 5 kHz. The fundamental frequency is used as a voicing detector; formant tracking is performed only in areas of the utterance for which there are fundamentals. The RMS parameter is a measure of the total energy from 0 to 10 kHz over each 10-msec. interval.

The frequencies of the spectral peaks below 5 kHz are extracted by the peak-picker from LPC spectra centered at each 10-msec. interval. The peak-peaker begins by building first-and-second-difference frequency tables. By inspecting these tables, the program locates all peaks and inflection points in the 0-5 kHz spectrum. If an isolated large-bandwidth peak is found, an off-axis spectrum is calculated in an attempt to resolve the peak into two peaks. If the total number of peaks and inflection points is greater than five, an off-axis spectrum is also calculated in an attempt to remove extraneous inflection points. Step 1 of Figure 2-7 shows the peak selections from the peak-picker.

The formant tracker begins by moving from left to right and linking frequencies of adjacent segments that are within a threshold difference of each other (the link is not made if a frequency in one segment could be linked to more than one frequency in the adjacent segment). Anchor areas are then established in which formant labels can be assigned unambiguously; this is done by examining sequences of three consecutive frames that contain three or more links. Ambiguity is detected whenever there is an extra peak or peaks or there is a missing peak. When F1 through F3 are extended to the right and left of the anchor, they are so extended only as long as the peaks are unambiguous. If an extra peak appears even for one frame, the extension of the anchor comes to a halt. Step 2 shows the anchor areas.

The remaining logic in the program is concerned with extending the anchors to the right and left into ambiguous areas and with establishing anchors in the areas where no unambiguous anchors could be established on the first pass. At this point, two kinds of context information are used as aids in resolving the ambiguities. Slope information based on formant movement from the anchor direction (either to the right or to the left) is used to help select the peak that best fits the past known formant slope. A search of the unknown area in the opposite direction from the anchor is made to determine whether a peak choice would continue to track. The one basic rule is that, whenever a possible low F1 peak appears for at least six frames, it is incontestably named F1, and the next higher or possible F1 is relegated to the slot "F4 or nasal formant" in the A-matrix. All frames tracked after the anchor stage are so indicated in the A-matrix, since they have less reliable formant information for segmentation and labeling. The final output of the formant tracker is shown in step 3. Step 4 shows the smoothed formant track.

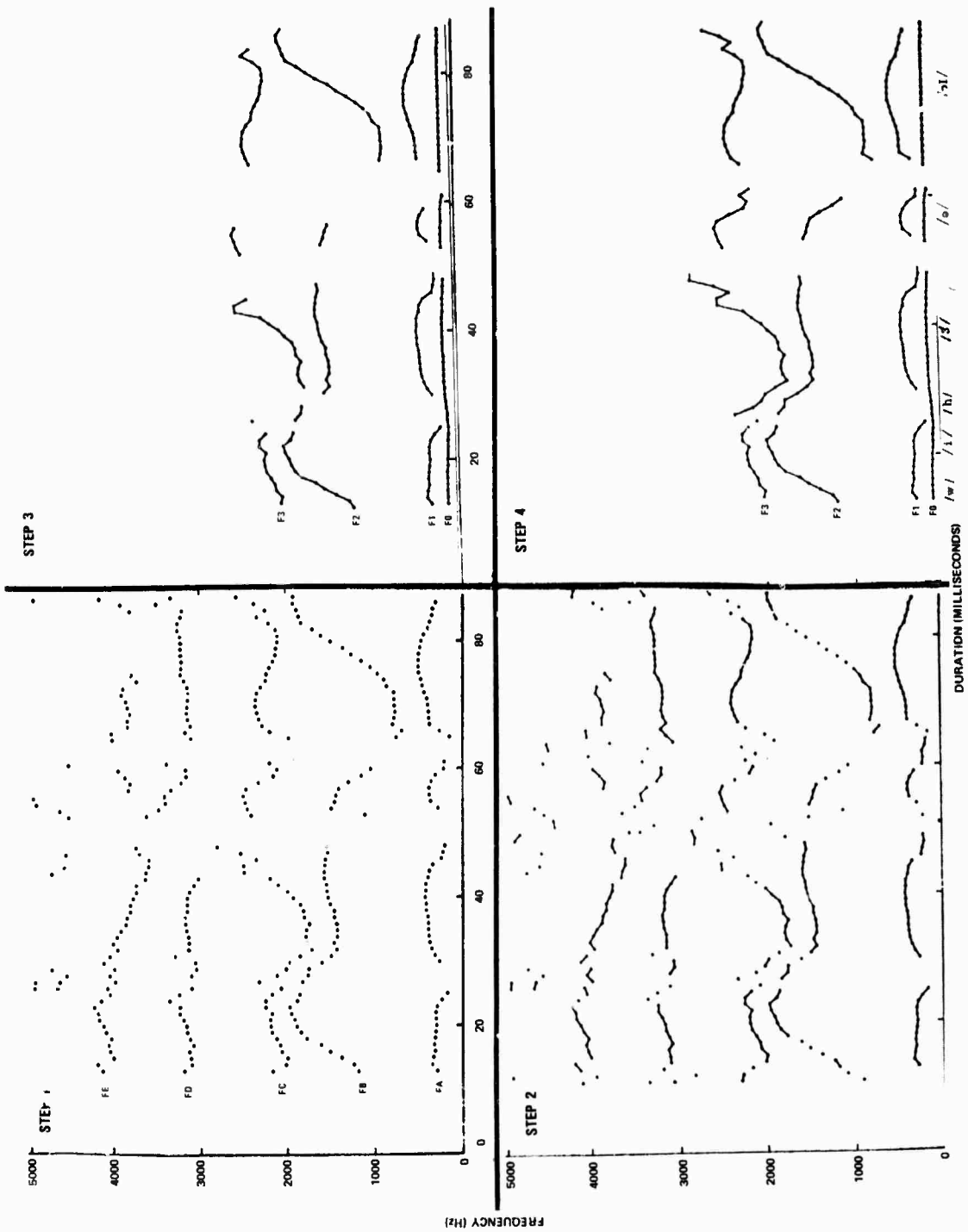


Figure 2-7. Example of Formant-Tracking Steps

Segmentation and Labeling

During the year, significant progress was made in the development and testing of our segmentation and labeling programs, which include programs for:

Syllable segmentation,

Acoustic phrase segmentation,

Acoustic stress and rate-of-speech analysis,

Vowel and sonorant analysis, and

Fricative and plosive analysis.

Developments in these areas are summarized in the following subsections. An example utterance, "What is the speed of it?", is used to illustrate the application of the various programs. At the end of this section, that utterance is displayed after the several programs have been applied.

Syllable segmentation. The primary importance of the use of the syllable as a phonetic unit stems from its use in a mapping strategy. In a good mapping strategy, it is important to be able to map units larger than phonemes or allophones since these relatively small units are influenced so strongly by their neighboring units. The syllable is a logical candidate for mapping, since it is just about the right length for a rhythm-based articulatory gesture and thus the unit within which most of the coarticulation should occur. Many other units could be used; for example, phonemes and various artificially induced units, such as 10-msec. frames. However, syllable boundaries tend to be more robust than phoneme boundaries. Syllables are genuine linguistic units (which assists the system in making a transition from the parametric representation to a linguistic representation). Moreover, syllables seem to provide natural breaks in the perception of continuous speech, as opposed to smaller units such as phonemes or allophones; indeed, Fujimura [20] has argued for the use of the syllable as a logical unit of speech recognition, largely upon the basis of the predictability of the concatenation properties of syllables, a property not shared by smaller, more traditional units of speech recognition.

A program that automatically segments a continuous speech utterance into syllables has been completed. Preliminary informal testing on speech from six male and two female speakers indicates that the program is about 95% accurate in isolating syllables. The program was adapted from an algorithm defined originally by Mermelstein [21] of Haskins Laboratories and was developed in collaboration with Mermelstein.

The program begins by dividing an utterance into so-called "voiced blocks", i.e., areas of contiguous 10-msec. voiced segments. (Voicing decisions are made by the pitch-tracking program described above.) Each voiced block contains one or more syllables. The program proceeds with the

following analysis in order to segment out the potential syllables. For each voiced block, a sonorant energy function is computed from each LPC spectrum in each 10-msec. frame in the block as follows:

$$SE_j = \sum_{i=1}^{100} w_i \cdot S_i \ ,$$

where $S_1,\ldots,S100$ are the first 100 spectral values in the LPC spectrum and where $w_1,\ldots,w_{100}$ are a set of weights designed to emphasize the portion of the spectrum that contains the major concentration of energy for vowels and sonorants.

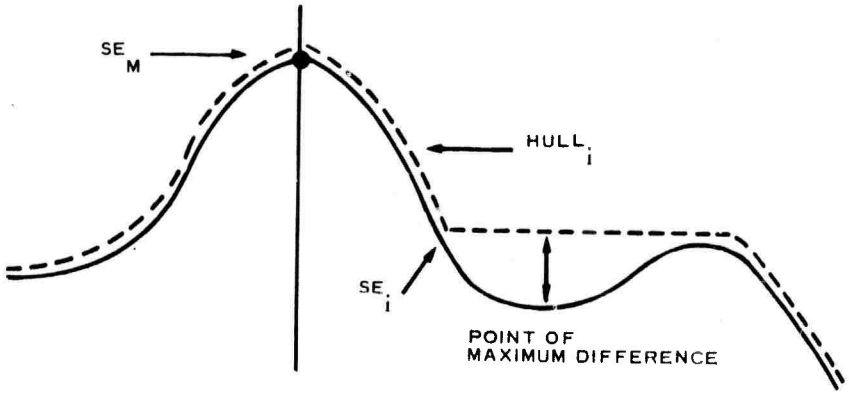
The next step is to examine each voiced block to determine whether it contains more than one syllable and, if so, to break it up into its component syllables. This is done by constructing a convex hull function from the sonorant energy function defined above. Briefly, the convex hull function ($HULL_i$) is defined as follows: Let SE_M denote the maximum value of the sequence $SE_0,\ldots,SE_n$. Then we move from left to right in defining $HULL_i$ by

$$\begin{cases} HULL_0 = SE_0 \\ HULL_{i+1} = \max \{HULL_i, SE_{i+1}\} \text{ for } i=0,1,\ldots,M. \end{cases}$$

Moving from right to left, we define

$$\begin{cases} HULL_N = SE_N \\ HULL_{i-1} = \max \{HULL_i, SE_{i-1}\} \text{ for } i=N,N-1,\ldots,M+1. \end{cases}$$

This convex hull function is monotonically nondecreasing from the start of the segment to its point of maximum loudness (i.e., SE_M), and is monotonically nonincreasing thereafter. A typical convex hull function is depicted as follows:



Within the segment, the point of maximum difference between the convex hull function and the sonorant energy function is considered to be a potential syllable boundary. The magnitude of the difference is a primary parameter in determining whether or not a syllable boundary exists. If his magnitude exceeds a preset threshold A, then the syllable break is made without further analysis. On the other hand, if this threshold is not exceeded, but a lower threshold B is exceeded, then the syllables and acoustic features in the A-matrix are examined to determine whether a syllable break has occurred. In examining the A-matrix, features such as the following are used:

1. Each syllable contains one and only one vowel.
2. The presence of a voiced "flap" (e.g., /d/) or a voiced dip signals the presence of a syllable boundary.
3. Each syllable must have a minimum duration.

This process is carried out recursively, so that if a boundary is found, the process is reapplied to both halves.

Since the last syllable boundary in an utterance is usually poorly articulated, thresholds A and B are lowered for this case. Moreover, the beginning of an utterance can also be problematical due to prevocalization, which produces a false syllable that the program eliminates based on its extremely low sonorant energy and short duration.

Figure 2-8 is an example of the processing of the utterance. "What is the speed of it?" The sonorant energy function is shown (only in voiced areas) along with vertical lines depicting the syllable boundaries. Note that the utterance contains six syllables and that the program automatically determined the same number. Listening tests indicate that the syllable boundaries are plotted in their proper positions.

Acoustic phrase segmentation. An acoustic phrase is a connected group of syllables having a simple pitch contour. During the contract year, a program that automatically segments an incoming utterance into acoustic phrases was completed.

Knowing the locations of the acoustic phrases in an utterance helps to determine acoustic stress, provides important clues to the syntactic complexity of an utterance, determines the presence of a pitch rise at the end of an utterance (which indicates the possibility of an interrogative sentence), and, since phrase boundaries are almost certainly also word boundaries, permits the parser to begin a new path when a given parsing strategy must be abandoned. Furthermore, the word-boundary information restricts the mapper's search for words to either side of the boundary.

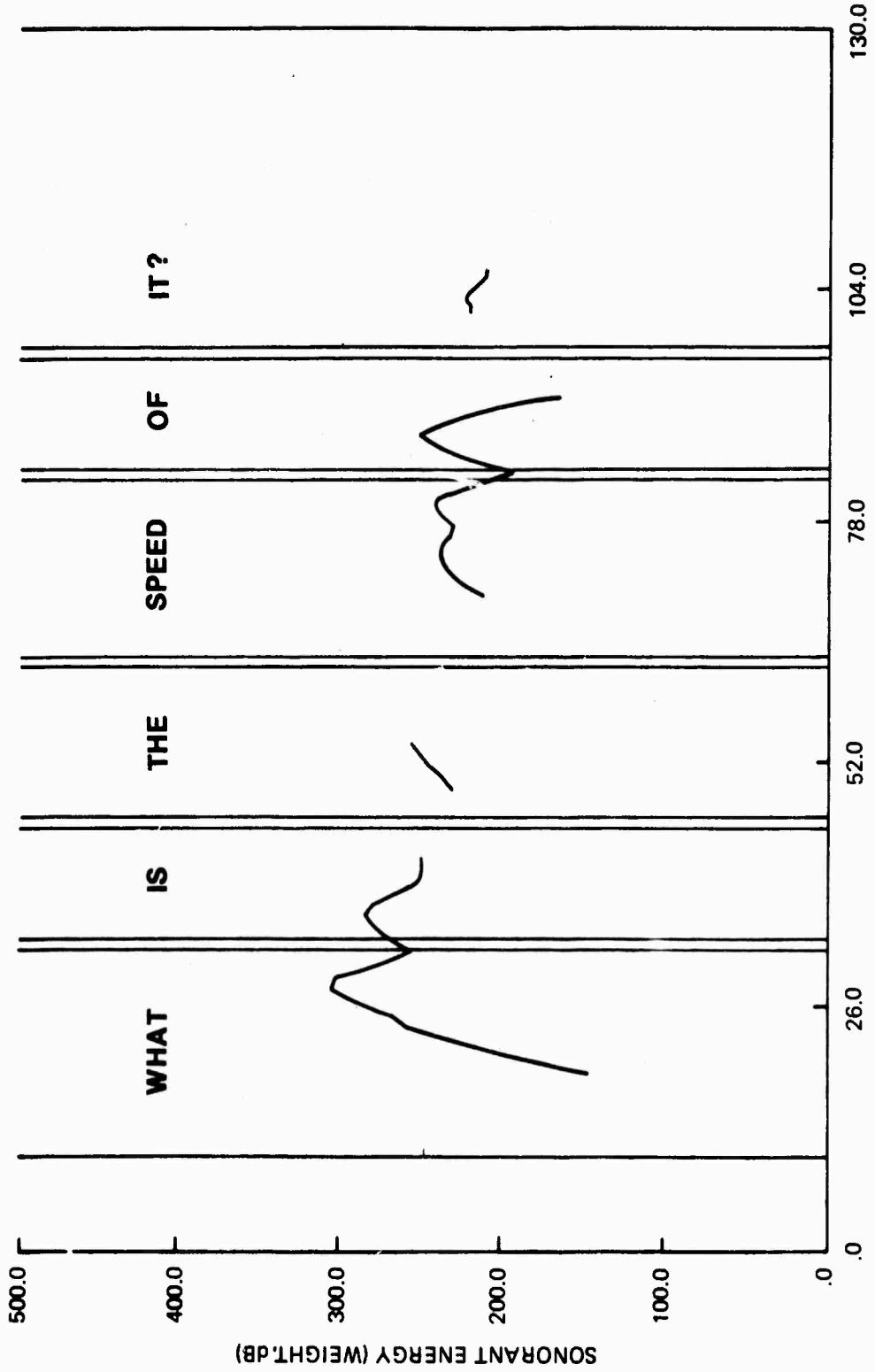


Figure 2-8. Automatic Syllable Segmentation of the Utterance "What is the speed of it?"

The pitch value in the center of the voiced portion of each syllable (determined by the syllable segmentation program) is used instead of the sonorant energy function used above. The convex hull function algorithm is then applied to this sequence of points. The point of maximum difference between the convex hull function HULL and the pitch contour PITCH is first determined. Next, if for this point

$$\frac{\text{HULL}}{\text{PITCH}} > A \quad ,$$

where A is a preassigned threshold, a phrase boundary is marked. The same procedure is then applied to the resulting two phrases if a phrase boundary was found. The program continues recursively until no further boundaries can be marked. Each time a boundary is located, it occurs within the voiced part of a syllable. This boundary is then moved to the end of the syllable having the lower pitch.

Phrase contours are labeled falling, rising, fall rise, or rise-fall based on a parabola least-squares fitted to the non-zero values of the pitch contour. The parabola is defined by

$$p(t) = at^2+bt+c$$

where $p(t)$ is the value of the pitch contour at time t. The extremum of the parabola occurs at time $PK = \frac{-b}{2a}$. Assume that the phrase occurs from t_1 to time t_2. Then eight possible cases can occur, outlined in the following table:

VALUE OF a	PK	LABEL
$a<0$	$PK \leq t_1$	Falling
$a<0$	$PK \geq t_2$	Rising
$a<0$	$t_1 < PK < t_2$	Rise-fall
$a>0$	$PK \leq t_1$	Rising
$a>0$	$PK \geq t_2$	Falling
$a>0$	$t_1 < PK < t_2$	Fall-rise
$a=0$	$b>0$	Rising
$a=0$	$b \leq 0$	Falling

Figure 2-9 is an example of the program applied to the utterance "The Seawolf has six torpedo tubes." The pitch contour is shown, along with the phrase boundaries (vertical lines), and the fitted parabolas. The labels for the pitch contours are "rise-fall," "rise-fall" for the two phrases. Figure 2-10 shows the result of the program applied to the utterance "What is the speed of it?"

Acoustic stress and rate-of-speech analysis. In the mapping strategy, it is important to know the acoustic stress of each syllable. There are three reasons for this. First, reduced vowels (primarily schwah) are distinguished more by their stress level than by their formant frequency structure. Second, in a "bottom-driving" strategy (in which words are located and recognized purely on the basis of acoustic clues), it is important to begin the bottom-driving with a stressed syllable, since this will contain more reliable acoustic-phonetic information than a syllable with a lower stress level. Third, agreement between predicted stress levels and machine-generated stress levels is a part of the scoring function of the mapper.

Three parameters are calculated for each syllable:

1. Duration (DUR) of the voiced portion of the syllable,
2. Intensity (I), defined to be the maximum RMS energy in the syllable,
3. Relative pitch (RP), defined by

$$RP = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} [F_0(t) - (at^2+bt+c)]dt \quad ,$$

where t_1 and t_2 are the beginning and end of the voiced portion of the syllable, $F_0(t)$ is the time-varying pitch over the voiced portion, and at^2+bt+c is the parabola fitted to $F_0(t)$ over the phrase as above.

The average value of the first two parameters ($\overline{DUR}$ and $\overline{I}$, respectively) is calculated over all syllables in the utterance. Stress is assigned by constructing a scoring function defined as follows: For a given syllable, if DUR $\geq \overline{DUR}$, then one "point" is assigned. If also $I \geq \overline{I}$, another point is assigned. If RP ≥ 0, then still another point is given. Thus, each syllable is assigned a score of 0, 1, 2, or 3. Stress levels are then assigned as follows:

Score	Stress Level
0	Reduced
1	Unstressed
2	Medium stress
3	Stressed

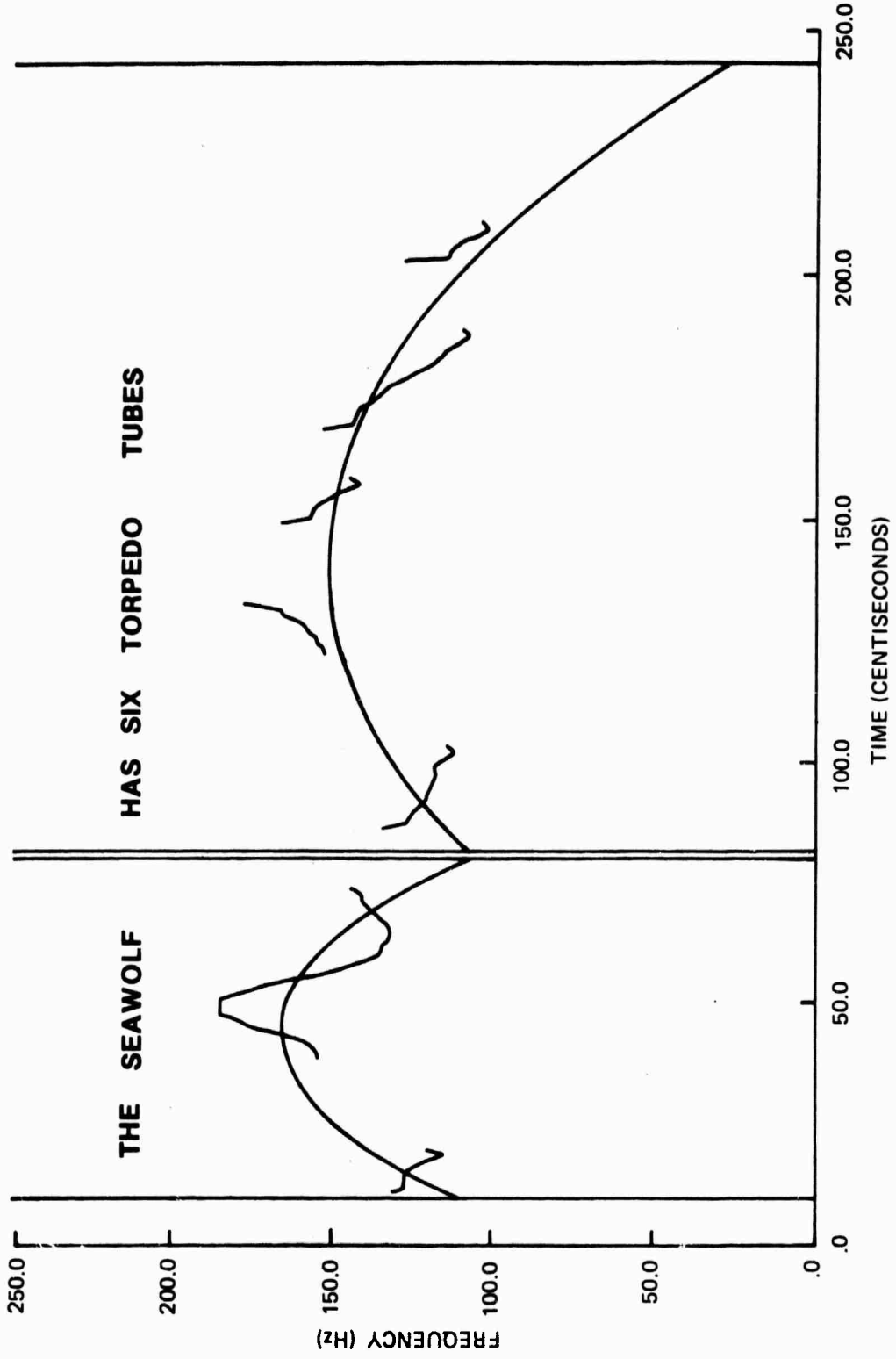


Figure 2-9. Automatic Phrase Segmentation of the Utterance "The Seawolf has six torpedo tubes."

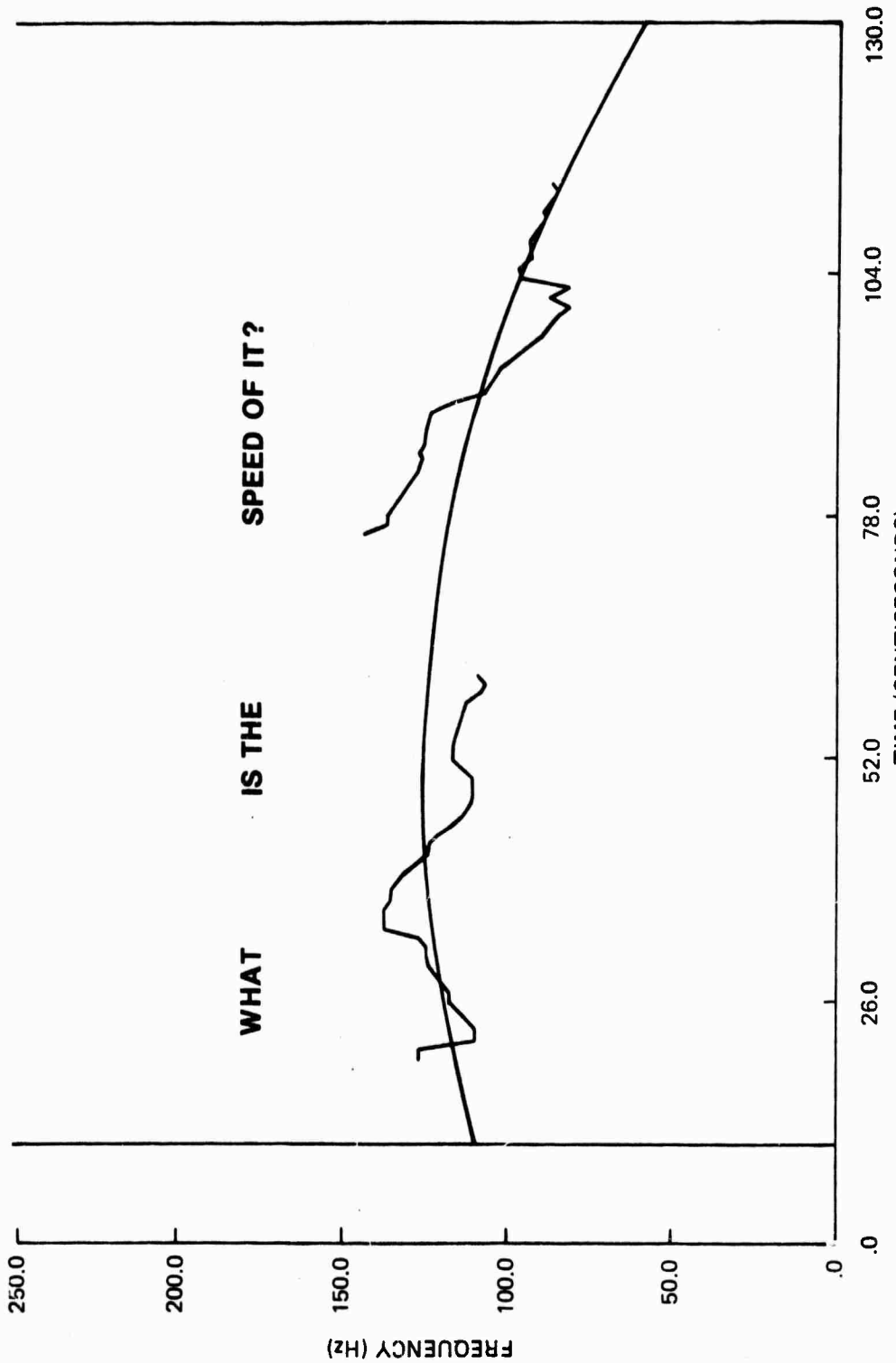


Figure 2-10. Automatic Phrase Segmentation of the Utterance "What is the speed of it?" (single phrase)

Since the last syllable in an utterance is generally lengthened and lowered in intensity, its stress is assigned to be medium stress or unstressed based soley on whether RP $\geq$ 0.

Another parameter important to a mapping strategy is rate-of-speech. Syllable segmentation is generally about 95% accurate in isolating syllables. When a word is hypothesized by the parser, the mapper first assumes that the machine-generated syllable boundaries are correct. A word match is attempted; if it fails, the failure may be due to the fact that the boundaries are misplaced. Given the rate-of-speech (defined to be the number of syllables per second), it is possible to remap the word using uniform syllable boundaries extrapolated from the rate-of-speech measurement. This parameter will also be useful in determining the applicability of fast-speech rules.

The program inverts the duration of each syllable for each 10-msec. frame, yielding a measure of syllables-per-second for each frame. These are smoothed, using a 100-frame moving average, and the result is inserted into the A-matrix.

Vowel and sonorant analysis. The main purpose of the vowel-sonorant (VOWSON) program is to locate steady-state segments and to enter segment boundary and label information into the A-matrix. Not all vowel-sonorant events are steady-state. The definitions of some events are, indeed, tied to the pattern movement of the formant frequencies; they make a gesture toward a target but do not attain the target or do not hold a fixed position for even a short period of time. Also, some events do attain a steady-state, but it may have been influenced by surrounding sounds and does not match closely to "pure" vowel or sonorant targets as indicated in the speaker-dependent tables. The results of a retroflexion experiment indicate algorithms for handling retroflexed vowels, but nasalized and lateralized vowels cannot be meaningfully handled until the appropriate experiments have been performed and the results interpreted.

The strategy of the present VOWSON program is to locate, segment, and label appropriately only those steady-state areas that it can handle with a high degree of reliability. All other voiced areas are left for the lexical mapping procedures to interpret in a syllable, word, or phrase context. VOWSON does provide the mappers with information extracted from the parameters to enable them to map more efficiently. This information is provided in the form of the following kinds of indicators:

1. Indications are made in the A-matrix as to discontinuities in F1, F2, or F3 based on the difference in formant frequencies between adjacent frames.

2. An appropriate rise or fall indicator is turned on for each frame in which the frequency of F2 change exceeds a threshold from one frame to the next. This enables the mappers to quickly discern slow-moving F2 changes from those that are moving more rapidly.

3. A sporadic voicing indicator is turned on if the fundamental frequency goes on and off over a contiguous period. This indicator is used as a flag to the fricative-plosive program to investigate the area.

4. A retroflexion indicator is turned on for all frames in which F3 is below a threshold value. The threshold value is defined as being half the F3 distance between /ɜ/ and /u/.

5. A lateralization indicator is turned on for all frames in which the F1, F2, F3 frequency pattern is within a threshold difference of the pattern given for /l/ in the speaker-dependent table.

6. A nasalization indicator is turned on when the F1 frequency is low and the F1, F2, F3 frequency pattern is not /i/-like or /u/-like.

7. Contiguous voiced areas not exceeding three frames for which formants are missing or erratic are labeled "voiced junk." They may be non-speech phenomena such as tongue clicks, glottal sounds, or portions of bursts.

8. A falsetto indicator is turned on for frames having an F0 greater than 350 Hz. A vocal fry indicator is turned on for frames having an F0 less than 65 Hz. ("Vocal fry" refers to what are often called "creaky voice" sounds.)

9. If the number of slope changes in the digitized signal exceeds a threshold for a voiced frame, a voiced fricative indicator is turned on.

VOWSON also detects energy dips in voiced areas and indicates the dip areas in the A-matrix. The parameter used for dip detection is the RMS after a three-point average smoothing has been performed. The technique used is similar to that described by Weinstein et al. [22]. Each minimum is tested against its surrounding maxima to ascertain that the ratio of the minimum to each surrounding maximum is within a threshold of .80, and that the combined ratios are within the threshold 1.20. The dip-location technique was applied to 69 utterances from a protocol. Some sample results are given below:

	Words or phrase (phoneme or boundary underlined)	# Times phoneme (or boundary) found correctly	# Occurrences of word (or phrase)
Detection of voiced plosives (/b/,/d/,/g/)	submarine(s)	18	21
	number	10	11
	submerged	7	11
	Albacore	1	1
	guided	3	3
	guided	1	3
Detection of unvoiced plosives (/p/,/t/,/k/)	what is	3	10
	Washington	3	9
	thirty	3	4
	computer	1	1
	subset on	1	1
Detection of morph boundaries	missile | launchers	3	14
	the | Ethan	2	5

Some other sounds labeled as dips were: of the Soviet, Lafayette, length, united, many.

VOWSON utilizes previously constructed speaker-dependent vowel-sonorant tables. These tables contain entries for the following ARPABET symbols: IY, IH, EH, AE, AA, AH, AO, OW, UH, UW, AX, ER, L, W. Each sound has F1, F2, and F3 frequency values associated with it. The frequency values for IX, R, L, and W are assigned by the program from existing sounds in the table. The F1 of IX is defined as half the distance between the F1s of IY and IH; the F2 and F3 of IX are defined as half the distance between the respective F2 and F3 values of IH and AX. The F1 of R is defined as 3/4 the F1 value of ER; the F3 of R is defined as the F2 of ER, and the F2 of R is defined as the F1 of R plus 60% of the distance between the F3 and the F1 of R. The F1 of L and W is defined as half the distance between the F1s of IY and IH. The F3 of L is defined as the F3 of IH. The F2 of L is .382 of the distance between the F3 and F1 of L. The F2 of W is 200 Hz less than the F2 of L and the F3 of W is 400 Hz less than the F3 of L. VOWSON also assigns frequency values to a group of retroflexed vowels: IY, EH, AH, OW, UW, ER. The algorithms used are those described in the results of a retroflexion experiment [23].

The F1, F2, F3 frequency values of the vowels in the speaker table are converted to a linear scale from 0-99. This allows matching to be done on the basis of linear distance rather than Euclidean distance, reducing computational costs. The conversion is made by finding the minimum and maximum F1, F2, F3 values from the table (excluding the retroflexed vowels) and extending these minima and maxima ±15%. The distances between the minima and maxima are then divided by 99 to yield the scale factor for each formant. Each frequency resonance can then be converted by subtracting the minima for that resonance and dividing by the respective scale factor. An example table for speaker WAB is shown in Table 2-4. Also shown are the F1, F2, F3 minima and maxima values and their respective scale factors. If a formant frequency is below the minimum frequency, its default setting is 0; if it is above the maximum, its default setting is 99. All F1, F2, F3 values in the A-matrix are converted to scaled values, and the scaled values are stored in the A-matrix as additional information.

The first phase of segmentation is to find nuclei within the utterance. Starting at the beginning of the A-matrix and proceeding to the end, each voiced (V) area is located and labeled. Voiced junk areas are ignored. The nucleus finder is run in all areas having the following characteristics:

1. The entire area is labeled V.
2. Each frame in the area has an F0, F1, F2, and F3.
3. The area does not contain a dip.
4. The area is ≥3 frames.

The first task of the nucleus finder is to locate the frame(s) of peak RMS energy in the defined V area. This is done by using the first-difference values between adjacent smoothed RMS values. (There may be more than one RMS peak if the area includes more than a single vowel surrounded by sonorants.) The other parameter used for nucleus finding is the absolute first difference in scaled F1, F2, F3 in adjacent frames for the defined area. If this value for all frames exceeds a threshold, then there is no nucleus, and segmentation and labeling are not attempted in that area. This is because the formant frequencies are moving too rapidly to define a steady-state area, and the problem of how to interpret the area is deferred to the mappers. If there are first-difference values below the threshold, the frame showing the smallest difference (least amount of change) is selected as the nucleus. If more than one frame has the same minimal difference, the frame closest to an RMS peak is selected as the nucleus.

Once the nucleus is defined, the segment boundaries are determined by moving to the right and left of the nucleus until a scaled F1, F2, or F3 value differs from that of the nucleus by more than a threshold value (one formant frequency outside the threshold is allowed for one frame), or until the beginning or end of an adjacent segment is encountered. More than one segment may be defined in the area if the undefined gaps between segments and/or the beginning and end of the area are greater than a threshold number of frames. The beginning, end, and nucleus indicators for each segment are entered in the A-matrix.

TABLE 2-4. VOWEL-SONORANT TABLE FOR WAB

Phone	Hz.			0-99 Scaling			
	F1	F2	F3	F1	F2	F3	
IY	273	2304	2851	8	85	81	
IH	429	1914	2695	39	66	73	
EH	526	1835	2598	58	62	68	
AE	625	1660	2343	78	53	55	
AA	645	1093	2440	82	25	60	
AH	585	1406	2539	70	40	65	
AO	645	1054	2617	82	23	69	
OW	507	1093	2382	54	25	57	
UH	429	1210	2382	39	30	57	
UW	351	1152	2246	23	28	50	
AX	546	1367	2304	62	38	53	
ER	400	1445	1640	33	42	20	
L	351	895	2695	23	15	73	
W	351	695	2295	23	5	53	
IX	351	1640	2499	23	51	63	
R	300	986	1445	13	19	10	
IY	351	2109		23	75		
EH	463	1952		46	68		Retro-flexed Vowels
AA	645	1073		82	24		
OW	457	993		44	20		
UW	390	1181		31	29		
ER	414	1679		36	54		

Formant	Minimum	Maximum	Scal. Factor
F1	233	741	5
F2	591	2649	20
F3	1229	3278	20

Labeling is done on the basis of the scaled F1, F2, F3 values found in the nucleus frame. Linear distances are computed from each vowel and sonorant in the speaker table, these distances are ordered, and the first four choices (closest matches) below 50 are selected and entered in the A-matrix. The score at the present time, is simply the linear distance of the match. If the difference between a single formant in the nucleus (either F1, F2, or F3) and the corresponding formant in a vowel or sonorant in the table exceeds a threshold, the vowel or sonorant is not acceptable as a choice, even if the composite score is less than 50.

Labeling proceeds as follows. If a segment is immediately preceded, within six frames, by two consecutive frames that have retroflexion indicators turned on, then the retroflexed IY from the speaker's vowel table is used instead of the non-retroflexed IY, and only the F1 and F2 distances are measured for the other sounds. Likewise, if the segment is followed within six frames by two consecutive frames that have retroflexion indicators turned on, then the retroflexed vowel table replaces the non-retroflexed vowel table. If the nasal indicator is turned on for the segment, then a NA (nasal) choice with a default distance of 50 is inserted as the last possible choice in the A-matrix. The default is used because the locations or effects of nasal formants and zeroes are not known at present. No special handling of vowel lateralization or nasalization is attempted at this time.

The nucleus finder, segmenter, and labeler are also run on dips if they exceed a threshold number of frames and all frames have an F0, F1, F2, and F3. If the dip is short, a default nucleus is defined to be the middle frame.

Fricative and plosive analysis. Major advances have been made in automatic acoustic-phonetic analysis of fricatives and plosives. Appropriate portions of the A-matrix are segmented and labeled P, T, K, B, D, G, HH, F, TH, S, SH, V, DH, Z, and/or DX by the program that performs this analysis (FRICPLOS). FRICPLOS is a continuation of work begun in 1973 on the application of linear prediction to the acoustic-phonetic analysis of unvoiced speech [24,25]. Our progress this year in this work has been extensive; the following are some of the highlights:

1. The fricative-plosive spectrum analysis process has been extended to yield useful spectra of voiced fricatives and plosives through the use of digital filter techniques. These sounds previously could not be analyzed successfully.

2. An efficient parametric representation of fricative-plosive spectra has been developed and extensively tested. It preserves acoustic-phonetically useful information while condensing each spectrum to five integers and two bits per A-matrix frame. Most usefully, the parameters enable numeric evaluation of how a fricative or plosive sound is changing with time.

3. A highly successful segmentation and labeling technique for fricatives has been developed. Frames in the A-matrix are grouped on the basis of spectral parameter stability, then labeled on the basis of spectral parameter average values within each group.

4. Reliable silence and plosive-burst-location algorithms have been develcped. Spectral parameters for the entire utterance are considered; the definition of silence adapts for each utterance. Bursts are located by amplitude pattern analysis.

5. An effective method for labeling plosives has been developed by combining detailed analysis of the plosive burst with information on its phonetic context. For each burst, a central collection of data is accumulated: burst spectral parameters for up to 40 msec. after onset, presence or absence of following retroflexion or lateralization, presence or absence of closure as evidenced by first-formant motion, voice onset time, formant frequencies at voice onset, presence or absence of preceding or following /s/, and other data. (Much of this data is based on preceding speaker-dependent analysis, the results of which have been placed into the A-matrix.) Independent burst-analysis routines then operate, each attempting to find its pattern in the data and thereupon label the burst. S-clusters (e.g. /ks/, /ts/, /st/, /ps/, /sp/) receive special consideration. Also taken specifically into account are plosive-sonorant coarticulation effects, such as occur in /tr/ and /kl/.

6. Techniques have been developed to make the voiced/unvoiced decision in labeling fricatives. (This is by no means a simple task in continuous speech, in which devoicing of "voiced" fricatives and overlap of voicing with "unvoiced" fricatives is common.) Information employed in making the decision includes the presence and exact extent of voicing, the presence or absence of preceding closure as evidenced by first-formant motion, duration of the fricative, and the presence of adjacent fricatives.

All the above techniques are included in the currently operational version of FRICPLOS. Well over a hundred continuous-speech utterances have been processed by FRICPLOS in performance testing.

Summary example. Figure 2-11 is an example of the utterance "What is the speed of it?" as processed by the phrase segmentation, syllable segmentation, vowel/sonorant, and fricative/plosive programs. The doubled slashes (//) indicate the phrase boundaries (only a single phrase exists in this case). Syllable boundaries are shown by double asterisks (**). The phoneme-label choices are the standard two-character ARPABET machine representation. The first line contains all of the first choices, the second line contains the second choices, etc.

```
4 -- WHAT IS THE SPEED OF IT?
SESSION NO.=    4       SAMPLE NO.=    4     SUBJECT NO.=15120

//  W AX DX ** IH  Z   Z **   V IX  S **  B  IY DX ** AX AX **   V IH   T ** //
// NA OW  B ** IX      V **  DH UW  Z **        B ** IX IX **  DH IX     ** //
//    UH DH ** UH     DH **     IH    **        DH **    J **     UH     ** //
//    AH    **              **            **           **    DH **     EH     ** //
```

Figure 2-11. Automatic Segmentation and Labeling of the Utterance "What is the speed of it?"

2.2.2.2 Lexical Mapping

The lexical mapping procedure, and the phonemic lexicon and its associated phonological processes, form the interface between the parser, which hypothesizes words and phrases, and the acoustic-phonetic data, in which the hypothesized words must be found. Various types of mapping capabilities are used, each designed to satisfy a particular requirement of the parsing strategy. The predictive mappers have a verification function; they attempt to give the parser a decision score as to whether a predicted entity actually could be present in a specified time region of the acoustic data. The predictive mappers include various "lookasides" for storage of prior mapping data, a short-word mapper, a word/phrase mapper, and a phone/cluster mapper. The subset mapper has a filtering effect; given some time point, it returns to the parser a small list of lexical items that the acoustic data suggest could begin at that point.

The phonemic lexicon, the central data structure for all of the mapping modules, contains the possible phonemic spelling variations a given word might have. These spellings are derived by the application of phonological rules to one or more root or "base" phonemic forms; they are then stored as a graph to minimize storage and processing time during mapping. The spelling graph of the word "submarine" is shown in Figure 2-12.

The Predictive Mappers

A predicted item may be a word or a phrase. Some time information is included in the prediction. Word boundaries are often imprecise, and one of the initial tasks in predictive mapping is to resolve time-boundary issues. In particular, phones overlap, or extensive coarticulation allows two words to merge together. Affixes tend to make word edges fuzzy. Pauses between words cause time gaps. These phenomena confuse the parser as to where to predict new items. The mapper attempts to reconcile predicted time information with what it already knows about mapped items. Time data may consist of either a boundary B (a specific time at which a word is hypothesized to begin or end) or a limit L (a minimum or maximum time at which a word can begin or end). Thus, there are four possibilities for boundaries: B-B, B-L, L-B, and L-L. The first three of these are the usual and expected forms; if an L-L search is called for, the mapper interprets this as a request for a bottom drive.

The predictive mapper begins by trying to eliminate requests with unreasonable boundaries. If the left boundary is greater than or equal to the right boundary, the mapper rejects the request. Similarly, if the request is too short or (with B-B) too long, the mapper eliminates those also. This check is made by reference to three factors: (1) the nominal length of the word as recorded in the lexicon (if there is more than one word in the request, the sum of lengths will be used), (2) an indicator of rate-of-speech, and (3) the pause structure of the utterance. Of course, if the requested word or phrase cannot be found in the lexicon, it is rejected. The high-level (word-and-phrase) lookaside is consulted to determine whether the requested word or phrase has been mapped

Figure 2-12. Stored Spelling Graph of the Word "submarine" Containing Alternative Pronunciations

before. If it has, the earlier mapping results are returned, and no further processing is required. If a word is determined to be a "short word," it is passed off to the short-word mapper. Otherwise, it is passed off to the regular phonological-rules pass to be mapped by the usual procedures.

Much information about previous mappings is stored in the lookaside memories. After making time adjustments, the mapper can see whether the predicted word or phrase has been predicted in the general time region before. If so, it can return this information without having to go through the actual exercise of mapping. A lookaside memory is a bi-directional array whose elements have a one-to-one correspondence with the 10-msec. frames in the A-matrix. Positive and negative results are stored here, the primary entry being an orthographic spelling or syntactic terminal name. (A lookaside memory is in fact a lattice, since it is possible for more than one lexical item to be mapped beginning at a given time frame.) Special routines exist to update and retrieve information from the two types of lookaside memory. The high-level (word/phrase) lookaside deals with words and phrases. When a word has been found with a high score, it is entered into the high-level lookaside memory. Also, if a word with reasonable time boundaries has been rejected, it too is entered into the high-level lookaside memory. The boundaries indexed in the high-level lookaside memory indicate where the word was found, rather than the boundaries given it by the parser. This increases the likelihood of a "hit." The purpose of this look-aside is to avoid duplication of effort: if the same request is made twice, we wish to repeat our first answer. Each entry in the high-level lookaside memory contains time boundaries, the orthographic spelling, the phonemic spelling, and the score of the word.

The syllable lookaside memory is used to save the mapper from remapping syllables. If a syllable is found with a good score, the result is saved in the syllable lookaside memory. Since it is possible to map more than one candidate in the same time region of the A-matrix, there is provision for storing a mapping score with each syllable. Modules that may update this memory are the predictive short-word and word/phrase mappers and the bottom driver. Two types of entries are provided: one for bottom-up (syllabary) information, and the other for top-down (phonemic) information. The syllable lookaside memory also provides a means of bottom-driving. If two consecutive syllables are found with high scores, the syllable-lookaside program subsets the lexicon to all words that contain those two consecutive syllables and requests a top drive on the subset.

The short-word mapper looks for all words that meet our definition of "short." Each lexicon entry is marked by hand as to whether it is short or not. The length of a word in phones or number of syllables, its syntactic behavior, and its acoustic characteristics influence this decision. We can expect that the number of "short" words will grow only slowly with vocabulary size, since most of them are common function words that all English subsets must use. The short-word mapper is heavily biased in favor of responding positively to the parser's hypothesis. Its primary function is to determine the length and location of the

short word. Short words include affixes; some short words have no vowel at all in the context of adjacent words. The short-word mapper tries first by looking in the syllable lookaside memory for its answer. If the answer is found there, no further processing is done. Otherwise, it will attempt a mapping. The spellings of the short words are found entirely in the lexicon and are not generated by rule. This is to account for the extreme variability of these words. The mappings generated are scored generously, but only those with a very definite result are recorded in the syllable lookaside.

The objective of the word/phrase mapper is to take a spelling graph from the phonological rules pass (see below) and try to map it. The mapping will take place in a left-to-right or right-to-left direction, but will not go back and remap from the beginning for every spelling variation. While this results in a slight loss of mapping power, a large savings in computation is achieved.

The mapping and scoring process consists of two coroutines: a scorer and a mapper. The scorer calculates syllable scores from the scores of the phonemes that make up each syllable. If the score for a syllable falls below a threshold, the syllable is pruned from the graph. If this causes the graph to become disconnected into two sub-graphs, the word is rejected. The mapper proceeds as follows: The graph is searched to find the set of first vowels. These are mapped, using syllable boundaries marked in the A-matrix to isolate position.

The mapper next returns and fills in any consonants preceeding the first vowels. The process is now repeated by locating the second vowels and returning to fill in consonants between the first and second vowels. If at any time a phoneme cannot be located, that phoneme is pruned from the graph; if this causes the graph to separate into two halves, the word is rejected. Finally, a word score is calculated from the surviving syllables by a full backtracking search of all possible syllable sequences. If the word cannot be mapped, the entire process is repeated using syllable boundaries extrapolated from the rate-of-speech.

The phone/cluster mappers ("sniffers") share a common calling sequence. They are parameterized in such a way as to use the results of the phonological rules to deal with duration variations, lateralization, nasalization, etc. The sniffers return a score (0-99) and boundaries that indicate the probability of a given phoneme in a given spot. They can look at the left and right context phonemes, if available. The sniffer scores are not normalized to necessarily return 99 at some time or another; they try to estimate probabilities and let processes at higher levels resolve those probabilities. Because of context sensitivity and cross-coarticulation, phonetic units may not correspond one-to-one with predicted phones. In some cases, a phonetic unit will be a phone sequence. A large number of acoustic-phonetic processes are incorporated into these low-level mappers. In general, they have access to all of the A-matrix information that is relevant to

short word. Short words include affixes; some short words have no vowel at all in the context of adjacent words. The short-word mapper tries first by looking in the syllable lookaside memory for its answer. If the answer is found there, no further processing is done. Otherwise, it will attempt a mapping. The spellings of the short words are found entirely in the lexicon and are not generated by rule. This is to account for the extreme variability of these words. The mappings generated are scored generously, but only those with a very definite result are recorded in the syllable lookaside.

The objective of the word/phrase mapper is to take a spelling graph from the phonological rules pass (see below) and try to map it. The mapping will take place in a left-to-right or right-to-left direction, but will not go back and remap from the beginning for every spelling variation. While this results in a slight loss of mapping power, a large savings in computation is achieved.

The mapping and scoring process consists of two coroutines: a scorer and a mapper. The scorer calculates syllable scores from the scores of the phonemes that make up each syllable. If the score for a syllable falls below a threshold, the syllable is pruned from the graph. If this causes the graph to become disconnected into two sub-graphs, the word is rejected. The mapper proceeds as follows: The graph is searched to find the set of first vowels. These are mapped, using syllable boundaries marked in the A-matrix to isolate position.

The mapper next returns and fills in any consonants preceeding the first vowels. The process is now repeated by locating the second vowels and returning to fill in consonants between the first and second vowels. If at any time a phoneme cannot be located, that phoneme is pruned from the graph; if this causes the graph to separate into two halves, the word is rejected. Finally, a word score is calculated from the surviving syllables by a full backtracking search of all possible syllable sequences. If the word cannot be mapped, the entire process is repeated using syllable boundaries extrapolated from the rate-of-speech.

The phone/cluster mappers ("sniffers") share a common calling sequence. They are parameterized in such a way as to use the results of the phonological rules to deal with duration variations, lateralization, nasalization, etc. The sniffers return a score (0-99) and boundaries that indicate the probability of a given phoneme in a given spot. They can look at the left and right context phonemes, if available. The sniffer scores are not normalized to necessarily return 99 at some time or another; they try to estimate probabilities and let processes at higher levels resolve those probabilities. Because of context sensitivity and cross-coarticulation, phonetic units may not correspond one-to-one with predicted phones. In some cases, a phonetic unit will be a phone sequence. A large number of acoustic-phonetic processes are incorporated into these low-level mappers. In general, they have access to all of the A-matrix information that is relevant to

them. Both the word/phrase mapper and the phone/cluster mappers have been completely flowcharted, and all coding has been completed for the phone/cluster mappers.

The Subset Mappers

Some progress has also been made in the construction of the subset mappers. Frequently, the parsing system needs to know what items are prime candidates for the next stage of predictive mapping. The subsetters provide a fast analysis of the A-matrix beginning at a given time frame, classify the phonetic patterns, and select items from the lexicon that belong to the classes. This provides a considerable reduction in the number of choices the parser must consider. The answer in this case is based on a bottom-up analysis of the A-matrix parameters. The subset may be performed either to the left or to the right of the specified boundary, depending on the form of the call. The subsetter also considers the possibility that the boundary has been shifted due to the mapper's "eating up" two identical phonemes in a row by accident. The lexical subsetter is used mainly by the parser, but it will also be possible to bottom-drive by doing a subset at the beginning of each phrase in the utterance, as determined by prosodics, and then top-driving on the most likely subclass. The analysis and classification techniques are identical to or compatible with those used in the bottom-driving module; some of the routines are common to both modules. The subsetters take advantage of any work already done by the bottom-driver by checking the appropriate data structures before beginning the processing.

Lexicon

The lexicon entries contain an orthographic spelling, a phonemic spelling graph, a nominal duration, bottom-up syllabary indices, and a short-word flag. The phonemic entry in the lexicon is a spelling graph that is constructed prior to system run time during compilation or during a pre-processing step. The graph allows alternative spellings to be mapped simultaneously; consequently, phonemic rule application results in a linear rather than an exponential increase in mapping time.

A specialist contractor, Speech Communications Research Laboratory (SCRL), has been actively assisting in the development of lexicons for the system. They have provided support in helping to develop base forms to be used for lexical entries. They have also been active in the related task of defining and evaluating rules for generating pronunciation variants from the base form. For part of this task they have used our phonological rules system.

Phonological Rules System

An early version of a phonological rules system was developed for generating variant pronunciations of lexical entries. It assumed that rules were applied in an unordered, optional manner. A different set of assumptions is required

for rule sets whose task is to derive inflectional or morphological endings. These rules are ordered and obligatory (if the context criteria are met), and successive rules operate on the output of the preceding ones so that only one spelling is derived. (These are the types of rules more often discussed in the linguistic literature.)

During this year, the phonological rules system was expanded to a much more generalized facility. It now provides for the building of lexicons and sub-lexicons. A lexical item may be tested individually or as part of a sub-lexicon. In this system, there are three types of rule-driver subroutines: ordered, unordered, and nondeterministic. Unordered and nondeterministic rule applications are very similar, the only difference lying in the fact that, in a small number of cases, a rule that would apply after a previous rule in a nondeterministic case would not apply in the unordered case because its left context was altered by the previous rule. The phonological rules system was designed as an independent rule-evaluation program. It has been slightly modified and incorporated into the mapper.

Lexical base-form spellings exist as properties of the orthographic words in a specially coded array structure. The phonological rules system makes use of this array coding during rule application; the result is that new coded arrays corresponding to variant spellings are produced. Under the old technique, the orthographic word was predicted, and its base-form property was extracted in the mappers. When a word can be predicted with one or more affixes, then the old approach is not adequate; the entire phonetic string must be derived and mapped as a whole. Routines were developed to construct new coded spelling arrays by copying one or more old ones. The mappers now receive as one input parameter a list of one or more words and/or suffixes. The spelling of each word is extracted; if a word has suffixes, they are derived using the ordered rule driver. The result is a single coded array, which may then be passed to the unordered rule driver for generating alternative pronunciations and mapping each one of them. This allows whole phrases to be mapped, with the added advantages that variants may be generated that result from applying coarticulation rules across word boundaries that are internal to the phrase.

2.2.3 System Hardware and Software

2.2.3.1 Digital Record/Playback Subsystem

A digital record/playback subsystem has been assembled for our PDP-11/40 computer based on experience with our Raytheon 704 computer system. As on that system, an amplified speech signal is digitized directly in real time with no intervening analog recording and is stored on a fast fixed-head disk. This recording process is reversed for playback. All data are moved using automatic direct-memory-access (DMA) hardware to allow high sampling rates; additional hardware assures unbroken continuity of sampling. The sampling rate is crystal-controlled for high absolute accuracy and long-term stability.

Speech enters the new system at high quality via an AKG condenser microphone or a Sennheiser headset-mounted microphone. The speech signal is amplified using low-noise, low-distortion equipment and is bandlimited by a 9,000 Hz low-pass filter (having 40 dB attenuation at 10,000 Hz) before being sampled at 20,000 samples per second.

Our experience with user variability has led us to employ a 14-bit analog-to-digital conversion system (Analogic AN5800); excellent digital recording can thus be obtained without any user gain adjustments. Employment of wide dynamic range in conjunction with a low-noise environment ensures good speech input during interactive discourse. The use of analog compression, limiting, or AGC circuits, whose unpredictable dynamic effects would complicate subsequent parameter extraction, is thus avoided.

A standard DEC DR11-B DMA interface provides block-transfer input/output for speech data to the PDP-11/40 Unibus. In order to ensure continuous sampling during the time required to reinitialize the DR11-B between block transfers, an SDC-designed controller provides a 64-word first-in, first-out buffer. This controller also includes timing and Analogic-to-DEC interface circuits.

2.2.3.2 Laboratory Facilities

A new physical facility has been designed and built to our specifications. It approximately triples our laboratory floor area. Included is an appropriate area for the PDP-11/40 and SPS-41 systems, an area for the IMP and 370/145 interface hardware that is sufficiently close to allow Local Host interfacing of the PDP-11/40 to the ARPANET, and a new IAC sound booth. The entire area was completed and in use by June.

2.2.3.3 Network Hardware Activities

In late 1973, an ARPA Network interface for the PDP-11 was developed at SDC. Designated the HSI-11A, this interface has been operational at SDC since January, 1974. In March, 1974, SDC was asked by the ARPA Interface Steering Committee (ISC) to submit HSI-11A for possible selection as a standard ARPANET interface for PDP-11 computers. In May, the HSI-11A design was selected. For several months thereafter, ISC members and the SDC staff conducted technical discussions, primarily by ARPA Network Mail, to specify an HSI-11B design suitable for production by some organization for widespread, general use on the ARPA Network. These discussions resulted in 11 engineering changes to the original HSI-11A design in order to meet ISC requirements. A documentation package on the HSI-11B was released in November, 1974 (Molho [26]). Also, SDC has assisted in prototype-building activities at Rand and BBN and has provided consulting services, as required. The prototype design has been forwarded to DEC at ISC request. The result of SDC's effort in this area will be that PDP-11 computers may be interfaced to the ARPA Network using reliable, off-the-shelf hardware.

2.2.4 CRISP

As research on the Speech Understanding System progressed, and as the size, complexity, and processing requirements became better defined, it became obvious that LISP or its derivatives (other languages are even less well suited) were not adequate to produce a system that could meet all of the research objectives. The most severe shortcomings were:

1. the extreme inefficiency of numerical computation (of which there is a large amount);

2. the inability to properly limit the scope, visibility, and access of names;

3. the inefficiency in saving, switching, and restoring processing context (a frequent occurrence);

4. representational limitations imposed by the available data structures;

5. constraints imposed on programs and data by address-space limitations; and

6. lack of formatted data output.

To remedy these deficiencies, a new programming system called CRISP has been developed that not only incorporates all of the capabilities of LISP but removes the constraining limitations and provides the missing capabilities. More specifically, CRISP:

1. produces object code that is efficient for both numerical and symbolic processing;

2. provides facilities for properly limiting the scope, visibility, and access to names and properties, permitting several people to cooperatively produce large complex programs with minimal housekeeping distractions;

3. efficiently saves, switches, and restores processing context;

4. provides generalized data structures, i.e., multidimensional arrays, n-tuples with repeating groups and elements, generalization on the two-pointer LISP node to nodes permitting from one to eight pointers, and functionals;

5. increases address space to a maximum of 16 megabytes and provides facilities for cooperating with virtual memory management;

6. produces formatted output, binary input/output, and general free-form input and output of any data structure;

7. provides the ability to freely mix infix, prefix, and machine-oriented language forms;

8. incrementally recompiles or batch compiles with the ability to redeclare data types in either case; and

9. provides means for modules in different virtual machines to communicate via the virtual channel-to-channel-adapter facility available in IBM's VM/370 system.

2.2.4.1 Present Status

During the latter part of 1974, the CRISP language and system were designed. The language design specification [27] was then published and distributed to potential users for comment and critique. In addition to presenting a semi-formal description of CRISP, the document attempts to illuminate the motivations for certain decisions and gives many example programs.

During the current year, large portions of the system have been programmed and debugged, and the following are operational: syntax analyzer, declaration mechanism, CRISP Assembly Language (CAP) assembler, I/O package, trigonometric functions, dynamic data structure allocators, and the context-of-evaluation primitives. The garbage collector has been programmed but has not yet been debugged. The detailed design of the compiler is nearing completion, and implementation has begun. The first usage of the system is the coding of the lexical mapper portion of the SDC-SRI system in CAP. As the compiler becomes available, portions of the mapper will be rewritten in CRISP, and the parser will be translated from LISP into CRISP.

The major technical obstacle in the implementation phase has been with the declaration mechanism. The specific issues were the handling of recursively defined types and allowance for redeclaration of the types of names without causing excessive recompilation. Satisfactory solutions have been found for both problems.

2.2.4.2 Technical Approach

One of our major technical goals is to program the system in its own language. This provides two important advantages: (1) the sophisticated user may access all parts of the system, code and data, to achieve capability extensions with relative ease, and (2) maintenance of the system is simpler (and significantly cheaper) because modifications can be made using the incremental assembler rather than regenerating the entire system.

Building the system in its own language implies the utilization of a bootstrap procedure. Our original intention along this line was to implement (in LISP) a compiler for a subset of the CRISP language. The approach we followed instead was to fully implement the CAP assembler in LISP, then hand-translate the assembler into its own language. Our reasons were: (1) needed parts of the system could not adequately be developed in higher-level language, (2) the assembler was needed as the final "pass" for the compiler, and (3) less effort was expended in recoding. As a result of the decision to implement the system in assembly language, CAP has been further developed than originally planned.

2.2.4.3 The System

Memory Allocation and Data Spaces

The heart of the CRISP system is the dynamic data allocator and memory management mechanism. The IBM 370/145 has a maximum address space of 16,777,216 bytes that is internally subdivided by CRISP into 4,096 quanta, each consisting of 4,096 contiguous bytes (and corresponding to the hardware page size used by VM). Memory is allocated in regions--a region is a set of contiguous quanta. A (data) space is a set of not necessarily contiguous regions. All data elements in the same data space are of the same kind.

There are three basic kinds of data spaces: static, selectable, and special. A static data space is completely allocated (but not necessarily filled) at system generation time. Its size does not vary dynamically during execution. An example of a statically allocated data space is the one-quantum area that holds character identifiers (identifiers with one-character print names). Other static data spaces are the pointer and numeric pushdown stacks (PDP and PDN, respectively), NAMEA, and NAMEB. One object in each data space is associated with each global name; the named object is the "value."

For a space to be selectable, it is necessary that more than one space of the same kind exist. An example of a selectable space is NODE2--a NODE2 object is the binary tree node of LISP created by the allocation function CONS. At any moment, one of the (possibly many) NODE2 spaces is selected. A use of CONS automatically allocates the new structure in the selected NODE2 space. The creation and selection of new spaces are easily accomplished using primitives provided in the system. Selectable spaces are valuable in many programs to overcome page thrashing. Specifically, if structures are built that will be heavily referenced at "nearly the same time" and they are placed in the same space, then, because of the increased likelihood that the structures will be on the same pages, the working set size will be decreased. Also, the garbage collector compacts (rather than building availability lists) so that structures remain in the space in which they were originally created. Further, spaces of the same kind may be merged into a single space.

There are three kinds of special spaces: IDENTIFIER, HEAP, and HANDLE. IDENTIFIER objects are hashed and singularized, i.e., there are never two identifiers in the system with the same print name. Therefore, the existence

of more than one identifier space is not meaningful and could, in fact, be harmful because singularity is used to support property objects for identifiers in a manner similar to LISP. HEAP spaces are used to allocate blocks of storage for specialized purposes. Associated with each process is a handle object (kept in the single HANDLE space); the handle contains the process's current status and its context of evaluation.

Input/Output Primitives

There are two general categories of I/O primitives: (1) file control and (2) data movers. The file-control primitives include: OPEN, SHUT, selection, and positioning. OPEN establishes a logical connection to a physical file through the operating system; SHUT severs such a connection. The possible media in which files may exist are disk, tape, terminal, card reader (spool), card punch (spool), printer (spool), and core (internally maintained by CRISP for intra-program communications). In the near future, it will also be possible to use files through virtual channel-to-channel adapters (CTCAs) provided by VM. This will make it possible for different virtual machines to communicate efficiently. The CTCAs also make it possible for a virtual machine to connect to the user TELNET as an ordinary I/O device.

The symbolic read and symbolic print primitives, respectively, operate on the current read and print "selected" files. When a symbolic input (or output) operation is initiated, the data are read (or written) from the file whose name is the value of the global variable RFILE (or PFILE). RFILE and PFILE may be rebound (and/or set) so that, as code blocks and processes are entered, exited, and resumed, the proper files are automatically selected.

The operation of the positioning primitives depends on the storage medium. The capabilities provided are: rewind, unload, skip file, backspace file, position at ith record in a file, continue a spooling operation, ease (purge) a file, write an end-of-file mark, and turnaround. (Turnaround is used to change the read/write direction of a file. For instance, turnaround of an output file does the equivalent of: write an end-of-file mark, backspace file, shut file, and reopen the file for input with the same line size, margins, etc.)

The data-moving I/O primitives are used to transfer information to and from files. Different primitives are used for binary and symbolic transfers. Binary transfers occur directly between heaps (or data structures containing no pointers) and files in a byte-for-byte serial manner, with no interpretation by the system. Symbolic transfers convert to (or from) an EBCDIC (or ASCII, if so specified) external representation. Any structure--including nodes, arrays, and n-tuples--may be read and printed symbolically; symbolic input is always "free-form." Symbolic output may be "ugly," explicitly formatted, or automatically "pretty" printed. Ugly printing outputs a structure as a token string--the only concession to legibility is that tokens are not unnecessarily split over line boundaries. The present, explicit format primitives allow data to be

printed left or right justified to a specified column. Future plans are for the inclusion of a format specification form similar to FORTRAN. Pretty-printing primitives automatically format the external representation of structures that do not fit on the current line. The technique we use is the standard one of using indentation to show structural nesting.

Process-Control Primitives

The process-control primitives provided by the CRISP system are a "parts kit" with which the user can fashion the set of control regimes that best serve his needs. The fundamental static units are blocks, functions, and processors. The fundamental dynamic unit is the process--an executing entity. A process may be in one of three states: active (presently computing), suspended (may be reactivated), or dead (may no longer be reactivated). Associated with each process is an object called a handle. The handle contains a process's complete internal and external states. The internal state is two pushdown stacks (a number stack and a pointer stack) that contain current variable bindings active in the process, return addresses to functions and blocks invoked in the process that have not yet exited, and the program counter. The external state contains, among other things, a context link and an abort link to other processes. The context link is used when global variables, not bound in a process, are referenced. When such a reference is made, a binding is searched for, through the chain formed by the context links.* To ensure termination of this searching procedure, a restriction is imposed--the processes considered as nodes and the context links considered as arcs must form a tree (no loops) with the NIL process as the root node. (The NIL process is the collection of all "top-level" variable bindings.) The abort link names the process that is to receive control if this process is aborted because of circumstances that cannot be handled internally. The processes considered as nodes and the abort links considered as arcs must form a tree (with NIL as the root node) so that a propagated error will not cycle indefinitely. The tree formed by the context links and the tree formed by the abort links need not be isomorphic.

Try-and-exit logic is also provided. UNWRAP is used to signal the occurrence of an unusual condition. The arguments to UNWRAP are: (1) the class of the condition causing the unwrap (there are sixteen possible classes; eight are reserved for system-detected occurrences such as I/O errors, and eight are left for user assignment), and (2) a message detailing the reason for the unwrap. UNWRAP aborts the present computation state by popping the stacks until an

*In actual operation, no searching is performed. All global variables are shallow bound for efficiency. The process control primitives are responsible for correcting the bindings whenever a new process is created or a suspended process is resumed. This tactic is adopted on the bet that context switching is a relatively infrequent occurrence compared to variable referencing.

active TRY is found.* The internal process state that existed when the TRY was entered is re-established. The TRY consists of several statements (expressions). The first is executed. If no unwrap occurs, then execution of the TRY is completed. Otherwise, the second statement is executed, and so on, until a statement is executed without the occurrence of an unwrap. If an unwrap occurs while the last TRY statement is executing, the unwrap is continued outward to the next active TRY. There is a TRY in the NIL process that will catch any unwrap that is not handled by an inferior process.

TRY and UNWRAP are extremely useful in two quite different contexts: (1) In "structured" programs, the occurrence of unusual (error) conditions is signaled by using UNWRAP. (2) If, in programs that use several algorithms in attempts to search for a solution, an attempted algorithm does not work, an UNWRAP returns control to the TRY, and the next algorithm is attempted.

2.2.4.4 The Language

In a programming language system such as CRISP, it is difficult to clearly distinguish language features from system features. This section will describe those features most often thought of as belonging to a language.

Language Formats

The CRISP system makes available to the user two basic languages: (1) CRISP--a high-level, procedural language, and (2) CAP--a machine-oriented language. Both languages are block structured and include a wide variety of data-structure-accessing primitives. Both languages share the same variable declaration and scoping mechanism, and CAP forms may be embedded into CRISP programs. Either language may appear in one of two formats: (1) Source Language (SL)--ALGOL-like with infix operators, or (2) Intermediate Language (IL)--LISP-like with Polish-prefix list structure. SL is ordinarily used as the programmer's language, and IL is used by programs that write or manipulate other programs.

Data Types

CRISP provides the user with a variety of atomic and non-atomic data types. The allowed non-atomic data types are nodes, arrays, and n-tuples. There are eight kinds of nodes: NODE1, NODE2...NODE8. A NODEi object has i ordered elements of type general. (NODE2s are the LISP binary node.) There are also eight "union" or multi-node types: MODE1...MODE8.

*A TRY specifies the class of error conditions that it is willing to accept. If the unwrap is for one of those specified conditions, then the TRY "catches" the unwrap. Otherwise, the TRY is bypassed and the search continues for another TRY. If no appropriate TRY exists in the currently active process, then the search continues into the process located by the abort link.

MODE8* = NODE8
MODE7 = NODE7 ∪ MODE8
MODE6 = NODE6 ∪ MODE7
MODE5 = NODE5 ∪ MODE6
MODE4 = NODE4 ∪ MODE5
MODE3 = NODE3 ∪ MODE4
MODE2 = NODE2 ∪ MODE3
NODEN = MODE1 = NODE1 ∪ MODE2

Thus, the type MODEi includes all nodes with at least i data fields. (Obviously, a node can be simulated with an array of general elements, but in many applications nodes are more natural. Also, because nodes are stored without any header information, they save four memory locations per occurrence.)

CRISP supports multi-dimensional arrays from 0 through 255 dimensions. Each dimension may have an extent of up to 32,767. An array type includes only the number of dimensions and the element type--not the extents. When an array is created (dynamically) the actual extents are specified. Extents specified in a declaration are used only as defaults in certain situations. When an array is created, its actual extents are stored in the header. The compiler always generates code that uses header information (rather than the default) to calculate element position from subscripts. (This tactic loses efficiency when constant subscripts are used to reference an array element but usually breaks even when all subscripts are non-constant expressions.) Array elements can be any kind of elements, even arrays.

The other kind of non-atomic type provided by CRISP is the n-tuple. An n-tuple comprises named elements and groups--a group is also a collection of named items and groups. Elements and groups may be repeated in much the same way as array elements. (However, the extents of repeats in n-tuples must be fixed at declaration time. If the extent(s) is not known, then the n-tuple element can be an array.) N-tuples are extremely useful because they provide a compact way of containing mixed-type data aggregates and because n-tuple references are highly mnemonic--thus improving a program's readability. Like array elements, n-tuple elements may be arrays or n-tuples. Also, an element that is an n-tuple may be flattened into its parent structure in order to conserve storage. (Normally, an element of type n-tuple is a pointer at the n-tuple; when flattened, there is no pointer, and the n-tuple is resident in place of the pointer.)

Variable Names and Scoping

In CRISP, there are two kinds of variable bindings--local and global. A local variable can be bound as an argument of a function or as a block variable. A local variable is visible only in the body of the function and in the interior of blocks lexically nested within the binding entity. The name of a local

*The usage of the word "mode" in this context should not be confused with its usage in ALGOL68.

variable is an identifier. In no case is a local variable visible outside the function containing its binding. A global variable can also be bound as an argument of a function or as a block variable. A global variable is visible in all places at which a local variable bound in the same spot would be visible. In addition, the binding of a global variable is visible in all function calls made within the variable's scope and in all processes that have the process containing the binding in their context chain. (This scoping strategy is called dynamic and is a replica of LISP's special-variable handling with a generalization to handle multiple processes). In addition to the dynamic bindings of a global variable that may exist, each global variable has a top-level binding (and value) in the NIL process. Thus, there is no such thing in CRISP as a reference to an unbound global variable.

Use of dynamic scoping of global variables (as opposed to the lexical, or static, scoping in ALGOL) has three major advantages. The most important is that it is possible to modify and recompile a small piece of a large program (e.g., a single function); it is not necessary to compile everything in the scope of the change, and incremental compiling (a function at a time) and interactive program testing are more efficient and more effective. A second major advantage is the ability to divide the programming load among several persons because they are able to produce separate lexical entities. Thirdly, programs can be organized in a more flexible manner because run-time decisions can be made on the binding set that is visible--in other words, the global context of evaluation can be computed.

The disadvantages of dynamic binding arise in large programs--problems arise when an intervening function call rebinds a variable used for communication between the "upper" and "lower" levels of evaluation. In general, when reading a program, it is difficult to determine what binding of a variable is being referenced. To help alleviate these problems, and other "name conflict" problems, CRISP provides a name-pool facility. All global names (variable names, function names, etc.) have a first and last name, each of which is an identifier. For example, the full name of the tangent function is TAN$CRISP. Its first name is TAN and its last name (or tail) is CRISP. In most CRISP programs, it is possible to reference or declare global entities by using only their first names; this is controlled by use of a default form. For example, assume that the following default is in effect.

```
DEFAULT XYZ(ABC,QRS);
```

All declarations and definitions that are not explicitly tailed receive the last name XYZ (the first argument of default). For instance,

```
DEC I INT, J$Q INT;
```

declares I$XYZ and J$Q to both be global integer variables. The second argument to default is used to tail identifiers that make free references (not lexically bound at the point the reference occurs). For example, suppose that the

identifier VAR occurs freely and that the above default is in effect. Then the compiler (assembler) first looks for a declaration of VAR$ABC--if none exists, then VAR$QRS is attempted.

By proper use of name pools--a name pool contains all entities with the same last name--most name-conflict problems disappear. A natural approach to the construction of a large program is to assign a unique tail to each module (collection of functions and declarations written by an individual that performs a set of related computations). Each module is compiled with a default that "sees" only its own name pool, appropriate system common pools, and the universal pool, CRISP (which holds all user-level functions and declarations provided by the CRISP system). Within each module, the (free) references to entities in any other modules not on the default list are explicitly tailed. (This is necessary because the pool name of the module is not on the default list of tails.) Another common tactic to avoid name conflicts is for a module to export (by use of explicit tails) those entry-point names and declarations that are documented as usable by others and to keep all other global names used by the module in its own pool.

Programming Example

Figure 2-13 shows a complete example of a program that traces a path through a maze. The purpose of the example is to indicate the flavor of the CRISP language and demonstrate several language features--it is not an example of an interesting or efficient algorithm.

2.3 PLANS

The major activity during the 1975-1976 contract year will be the integration, testing, and demonstration of the five-year system. The system will have a vocabulary of 1,000 words and will allow many speakers of the general American dialect to maintain a dialogue with a data management system with reference to attributes of warships of the US, USSR, and UK. Acoustic feature extraction and acoustic-phonetic processing will be performed on the PDP-11/40 and SPS-41 computers. All subsequent processing, such as that required for parsing, semantics, pragmatics, and word verification, will be done on the IBM 370/145 computer, connected to the PDP-11/40. Programming on the PDP-11/SPS-41 configuration will be done in FORTRAN, PDP-11 assembly code, and SPS-41 machine code. The use of CRISP on the IBM 370 will greatly enhance the efficiency of the higher-level processing.

In addition to supervising the integration and testing of the system, SDC will also conduct research, development, and refinement of the acoustic-phonetic processor, the word-verification procedures, and the prosodic-analysis functions. Capabilities of the acoustic-phonetic processor will be enhanced with the addition of acoustic-phonetic rules to handle vowel lateralization. Moreover, research will be conducted to determine a set of rules to handle nasal murmurs and nasalized vowels. Specifically, some recent results of Kopec et al. [28] and Mermelstein [29] will be implemented in computer programs,

```
DECLARE point<name ID,                    %'name of point in maze
            path ARRAY(*) point>,         %'set of reachable points
        visits NODE2,                     %'path to-date
        end point;                        %'end point of search


%'Findpath attempts to locate a path through the maze from the point,
%'beg, to the point, end.  If a path is found, it is printed and find-
%'path returns true.  Otherwise, no printing is done and findpath
%'returns false.  The try-exit logic is used for communication.  Unwrap
%'category 1 is used when a circular path is encountered and unwrap
%'category 2 is used to indicate success--the path is returned as the
%'unwrap message.  For simplicity, it is assumed that there is at least
%'one path away from each point "in" the maze.


FUNCTION findpath BOOL(beg point, end GLOBAL point)
   TRY 1^^2 (BEGIN visits GLOBAL:=NIL;
                   trypath(beg);
                   END,
             IF UNWRAPMASK=2 THEN (PRINT(UNWRAPMSG),RETURN TRUE)
                ELSE RETURN FALSE);


%'Trypath first checks for a circular path.  If found, the trouble is
%'reported by a category 1 unwrap.  If the current point is the end
%'point, then the good news is reported by a category 2 unwrap. Else,
%'each path away from p is tried.  All except the last such attempt is
%'protected by a try.  On the last attempt, failure is rippled upward
%'to a spot where an alternative is yet to be tried.


FUNCTION trypath NOVALUE(p point)
   IF p_name IN visits THEN unwrap(1)
      ELSE BEGIN visits GLOBAL:=p_name#visits;
                 IF p=end THEN UNWRAP(2,REVERSE(visits));
                 FOR i:=1 TO ARLN(p_path)
                     DO TRY 1 (trypath(p_path[i], NIL);
                        END;
                 trypath(p_path[ARLN(p_path)]);
                 end;
```

Figure 2-13. Sample CRISP Program to Trace a Path through a Maze

which will form the basis of a number of experiments designed to study nasals and nasalized vowels. Research on isolation and characterization of fricatives and plosives will continue, with an emphasis on the use of formant frequency trajectories from a plosive into a vowel to enhance recognition accuracy. Specifically, a voiced plosive will be labeled on the basis of both its release and frication properties and via an analysis of its formant-frequency transitions into the following vowel or sonorant.

Early in the 1975-1976 contract year, a 400-word extension of the 600-word Milestone System will be decided upon, and work will be initiated to develop a set of base forms for the resulting 1,000-word vocabulary. A vocabulary of this size will require major portions of the lexicon to be encoded in terms of morphs and their affixes, in order to avoid excessive storage requirements, since so many words can occur in a multitude of forms, as for example, "do," "does," "doesn't." Some limited use of derivational phonology rules will be made in the Milestone System. The five-year system will feature an expanded use of such rules.

An important extension to the present lexical matching procedures will be the development and use of analysis-by-synthesis techniques, also known as parametric mapping. For these types of techniques, a set of formant-frequency trajectories will be hypothesized for a predicted word or phrase. Using time-warping techniques, these formant trajectories will be adjusted to synchronize with the formant trajectories in the A-matrix, and a comparison will be made between these two sets of trajectories to determine the existence of the predicted word or phrase. This procedure is expected to yield better results than current mapping procedures, since it will be based on parametric, rather than phoneme-label, matches. This will also allow us to incorporate all of the theory of synthesis-by-rule and use it in the mapping procedure.

Some limited bottom-driving techniques will be used in the Milestone System. These will be based on the use of syllabic segmentation described earlier. The techniques will be refined and extended for use in the five-year system.

2.4 STAFF

Dr. H. Barry Ritea, Project Leader
James A. Balter (System Programming)
Jeffrey A. Barnett (CRISP)
William A. Brackenridge (Parameter Extraction)
Richard A. Gillmann (Lexical Mapping)
Iris Kameny (Acoustic-Phonetics)
Dr. Peter Ladefoged (Acoustic-Phonetics Consultant)
Lee M. Molho (Acoustic-Phonetics and System Hardware)
Douglas L. Pintar (CRISP)
Dr. Georgette Silva (Protocol Analysis and Lexicon Construction)
Rollin V. Weeks (Lexical Mapping)

2.5 PUBLICATIONS AND REFERENCES

[1] Denes, P. B., "On the Statistics of System English," JASA 35(6):892-904, 1963.

[2] Shoup, J., "Phoneme Selection for Studies in Automatic Speech Recognition," JASA 34(4):397-403, 1962.

[3] Paxton, W. H., and A. E. Robinson, System Integration and Control in a Speech Understanding System. Technical Note 111. Menlo Park: Stanford Research Institute (Artificial Intelligence Center), September, 1975.

[4] Gold, B., "Computer Program for Pitch Extraction," JASA 34(7):916-921, 1962.

[5] Miller, N. J., "Pitch Detection by Data Reduction," Proceedings of the IEEE Symposium on Speech Recognition, 1974, pp. 122-130.

[6] Reddy, D. R., "Fitch Period Determination of Speech Sounds," CACM 10(6), 1967, pp. 343-348.

[7] Noll, A. M., "Short-Time Spectrum and 'Cepstrum' Techniques for Vocal-Pitch Detection," JASA 36(2):296-302, 1964.

[8] Noll, A. M., "Cepstrum Pitch Determination," JASA 41(2):293-309, 1967.

[9] Ritea, H. B., The Cepstrum, the Cepstrally Smoothed Log Spectrum, and the Chirp Z-Transform, System Development Corporation Report No. TM-4857/200/00, 1972.

[10] Markel, J. D., "Automatic Formant and Fundamental Frequency Extraction from a Digital Inverse Filter Formulation," Proceedings of the IEEE Conference on Speech Communication and Processing, 1972, pp. 81-84.

[11] Markel, J. D., "The SIFT Algorithm for Fundamental Frequency Estimation," IEEE Transactions on Audio and Electroacoustics AU-20, 1972, pp. 367-377.

[12] Aschkenasy, E., M. R. Weiss, and T. W. Parsons, "Determining Pitch from Fine-Resolution Spectograms," Proceedings of the IEEE Symposium on Speech Recognition, 1974, pp. 288-289.

[13] Harris, C. M. and M. R. Weiss, "Pitch Extraction by Computer Processing of High-Resolution Fourier Analysis Data," JASA 35(3):339-343, 1962.

[14] Moorer, J. A., The Optimum-Comb Method of Pitch Period Analysis in Speech, Stanford Artificial Intelligence Laboratory Memo AIM-207, 1973.

[15] Skinner, T. E., "Autocorrelation Method for Determining Fundamental Frequency," Univac Intercommunication, April 23, 1973.

[16] Gillmann, R., "A Fast Frequency Domain Pitch Algorithm," JASA 58 (Supplement No. 1):S62(Abstract), 1975.

[17] Skinner, T. E., "Fundamental Frequency Determination - Analysis of an 'Octave Error,'" Univac Intercommunication, April 25, 1973.

[18] Markel, J. D., Formant Trajectory Estimation from a Linear Least-Squares Inverse Filter Formulation, Monograph No. 7., Santa Barbara: Speech Communications Research Laboratory, Inc., October, 1971.

[19] Kameny, I., W. A. Brackenridge, and R. Gillmann. "Automatic Formant Tracking," JASA 56 (Supplement):S28 (Abstract), 1974.

[20] Fujimura, O., "Syllable as a Unit of Speech Recognition," Proceedings of the IEEE Symposium on Speech Recognition, April 1974, pp. 148-153.

[21] Mermelstein, P., and G. M. Kuhn, "Segmentation of Speech into Syllabic Units," JASA 55 (Supplement):S22 (Abstract), 1974.

[22] Weinstein, C. J., S. S. McCandless, L. F. Mondshein, and V. W. Zue, "A System for Acoustic-Phonetic Analysis of Continuous Speech," IEEE Transactions on Acoustics, Speech and Signal Processing ASSP-23(1), 1975, pp. 54-67.

[23] Kameny, I., "Comparison of Formant Spaces of Retroflexed and Non-retroflexed Vowels," IEEE Transactions on Acoustics, Speech, and Signal Processing ASSP-23(1), 1975, pp. 38-49.

[24] Molho, L. M., "Automatic Recognition of Fricatives and Plosives in Continuous Speech Using a Linear Prediction Method," JASA 5(2):411 (Abstract), 1974.

[25] Molho, L. M., "Automatic Recognition of Fricatives and Plosives in Continuous Speech," Proceedings of the IEEE Symposium on Speech Recognition, April, 1974, pp. 68-73.

[26] Molho, L. M., HSI-11B--An ARPANET Interface for PDP-11 Computers, System Development Corporation Report No. TM-5434/000/00. October, 1974.

[27] Barnett, J. A., and D. L. Pintar, CRISP: A Programming Language and System, System Development Corporation Report No. TM-5455/000/00 (Draft), 31 December 1974.

[28] Kopec, G. E., A. V. Oppenheim, and J. M. Tribolet, "Speech Analysis by Homomorphic Prediction," JASA 58 (Supplement No. 1):S97 (Abstract).

[29] Mermelstein, P. "On Detecting Nasals in Continuous Speech," JASA 58(Supplement No. 1):S97 (Abstract).

3. LEXICAL DATA ARCHIVE

3.1 INTRODUCTION

The Lexical Data Archive (LDA) project addressed itself to the task of providing the ARPA Speech Understanding Research (SUR) projects with semantic and syntactic data for the words in their lexicons. The project sought to provide the following services for each SUR project: monitor a variety of lexical data sources, select the data having potential payoff for speech understanding, format those data for archival purposes, and provide for their dissemination to the appropriate SUR projects. The data in the archive are centered on the 3,000 or so words appearing in the early lexicons used by the SUR projects at Bolt Beranek and Newman Inc., Carnegie-Mellon University, and System Development Corporation.

3.2 PROGRESS AND PRESENT STATUS

The data archive, called the Semantically Oriented Lexical Archive (SOLAR), was designed during the 1973-1974 contract year. The methodology of construction decided upon was then implemented. Files with a significant amount of high-quality data became accessible via the ARPA Network, and data were distributed upon request to more than 65 researchers across the nation and abroad. Implementation was begun for the first eight of the following ten files:

1. A word index, which allows a user to easily determine the words for which data are being collected and the types of data currently available for a given word.

2. A bibliographic reference file, intended primarily as a resource for accessing the literature.

3. A file of semantic analyses, which contains formal treatments of the semantic properties of individual words as found in the literature.

4. A file summarizing the theoretical backgrounds of the technical documents from which the semantic analyses have been extracted.

5. A file explaining and commenting on the semantic components used in the semantic analyses.

6. A file of integrative summaries of conceptual analyses given in the literatures of philosophy and artificial intelligence for notions coinciding with or underlying the semantic components.

7. A file of collocational information extracted from definitions in Webster's Seventh New Collegiate Dictionary (W7).

8. A keyword-in-context (KWIC) file containing every context of each SUR word as found in the W7 definitions, in the Brown Corpus, and in selected speech dialogues.

9. For each SUR lexicon, a subfile of definitional links between words within that lexicon.

10. A file of semantic fields, designed for each SUR word by tying to it words found in certain definitional, synonymitive, and antonymitive relationships in W7, Webster's New Dictionary of Synonyms (WNDS), and/or Roget's International Thesaurus (Roget).

From the start of the period covered by this report, anticipating early completion of the project, we concentrated on checking out the programs required for constructing SOLAR's Definitional Expansions File (see pp. 307-309 of [1]), writing user's guides to the existing SOLAR files, and refining the ARPA network interface with those files. By midyear, the Definitional Expansion programs had been successfully run on test data, and user's guides to the Semantic Component and Conceptual Analysis Files [2,3] had been added to the four previously prepared user's guides [4,5,6,7].

The SOLAR data files have been stored on magnetic tape for use by linguists, researchers in artificial intelligen e, and philosophers. A paper explaining how to access the SOLAR files that had been submitted to the American Journal of Computational Linguistics was withdrawn so as to convert it into an account of what was learned in the course of building SOLAR. Meanwhile, the 23 integrative summaries now entered in the Conceptual Analysis File are being reformatted as an appendix to a report on the construction of that file [8] that will soon be submitted to The Philosophy Research Archive.

3.3 STAFF

Dr. Timothy C. Diller, Project Leader
Thomas Bye (part-time)
Enrique Delacruz (part-time)
Frank Heath (part-time)
John Olney (Consultant)
Nathan Ucuzoglu (part-time)

3.4 PUBLICATIONS AND REFERENCES

[1] Diller, T., and J. Olney, "SOLAR (A Semantically Oriented Lexical Archive): Current Status and Plans," Computers and the Humanities 8(5-6):301-312, September-November, 1974.

[2] Diller, T., T. Bye, and J. Olney, User's Guide to the SOLAR Semantic Component File. System Development Corporation Report No. TM-5292/003/00, July, 1975.

[3] Olney, J., E. Delacruz, T. Diller, and N. Ucuzoglu, User's Guide to the SOLAR Conceptual Analysis File, System Development Corporation Report No. 5292/004/00, June, 1975.

[4] Bye, T., T. Diller, and J. Olney, User's Guide to the SOLAR Semantic Analysis File, System Development Corporation Report No. TM-5292/001/00, April, 1975.

[5] Diller, T., User's Guide to the SOLAR Bibliography File, System Development Corporation Report No. TM-5292/000/01, December, 1974.

[6] Diller, T., and T. Bye, User's Guide to the SOLAR Theoretical Backgrounds File, System Development Corporation Report No. TM-5292/002/00, April, 1975.

[7] Diller, T., and F. Heath, User's Guide to the SOLAR KWIC File, System Development Corporation Report No. TM-5292/008/00, May, 1975.

[8] Olney, J., E. Delacruz, T. Diller, and N. Ucuzoglu, "A Partial Integration and Formalization of Conceptual Analyses given in the Recent Philosophical Literature for 23 Notions" (in preparation).

4. COMMON INFORMATION STRUCTURES

4.1 PURPOSE AND BACKGROUND

4.1.1 Goal

The need to share data for multiple applications, and the need to move existing data bases to new systems, make general techniques for data-base conversion desirable. These needs are especially apparent when the data are created and manipulated by increasingly complex data management systems.

The goal of the Common Information Structures project has been to develop techniques for data base conversion that can be applied to both existing and future data bases. It is assumed that the data bases are typically created by a data management system (DMS) that uses the operating system functions available on a particular hardware/software system. This is not to exclude sequential files that are created by special-purpose programs (rather than a DMS). Our purpose is to be able to convert and restructure a source data base into a newly defined target data base using generalized data conversion tools.

4.1.2 History of Research

The difficulties in converting a data base arise from the fact that data base structures are system (including hardware) and application dependent. Data bases are organized in the computer in ways that reflect different efficiency requirements, such as response time, storage space, and total cost. The organization of a data base can be viewed from three levels:

1. the logical level, which involves the description of field types, the grouping of fields into groups, and the relationships between groups;

2. the storage level, which involves access paths, inversion on data fields, and indexing mechanisms; and

3. the physical level, which depends on physical devices used and record/block organization of data on them.

Accordingly, two data bases, having the same logical organization, could be implemented in different DMSs and on different hardware, and would consequently have different storage-level and physical-level characteristics.

The conventional method of converting data bases for new applications is to write a special-purpose conversion program for each data base. The programmer who does this must know the storage-level and physical-level characteristics of the particular DMSs involved in great detail. Another, more general, approach is to define data description languages for all three structural levels, then specify in these languages the structures of the source and target data bases,

as well as the conversion statements [1-6]; discussions of this approach are presented in [7,8,9]. The necessary data description languages are complex, detailed, and difficult to learn and to use because they involve information at all three levels. In addition, because the data must be converted from the source physical environment to the target physical environment, implementation is complicated.

In examining the existing approaches, we concluded that another approach would more likely lead to ease of use and simpler mechanisms. This is the common information structures approach that we have developed over the past two years. This approach rests on the assumption that the data base conversion process can depend on conversion at the logical level to a maximal degree. Just as high-level programming languages are intended to divorce the structural and functional properties of programs from specific physical environments, we needed data-base conversion mechanisms that will move data in and out of specific physical environments. This can be achieved by using the existing query and generate capabilities of DMSs, which move data from their physical representation to the logical level and vice versa. Once the data and their relationships are represented logically, they can be restructured and manipulated with no reference to any storage-level or physical-level characteristics.

There is, of course, a trade-off between using the logical-level approach and using the approaches that have previously been proposed. It is between the need to deal, in the translation process, with many different formats (of input and output data to DMSs) in the logical-level approach, and the need to deal, in the other approaches, with the different internal data structures at the storage and physical levels. We believe that eliminating the complexities of storage and physical data structures from the conversion process far outweighs the complexities of dealing with different data formats. Moreover, our approach simplifies the languages required for specifying conversions, thus enhancing the ability of unsophisticated users (by whom, in this context, we mean applications programmers as distinguished from system programmers) to specify data-base conversions relatively easily.

As shown in Figure 4-1, the conversion system has three principal components: (1) a source reformatter, which reformats the output of the source DMS into a predefined standard data form (the standard form is an internal representation of data values and their relationships to achieve high efficiency in the translation process); (2) a translator, which logically restructures the data from the source standard form to a target standard form; and (3) a target reformatter, which reformats the target standard data into an input data stream for the generate facility of the target DMS. The reformatting process does not involve any logical restructuring of data, but is a one-to-one mapping of data values. The data translator operates only on logical data.

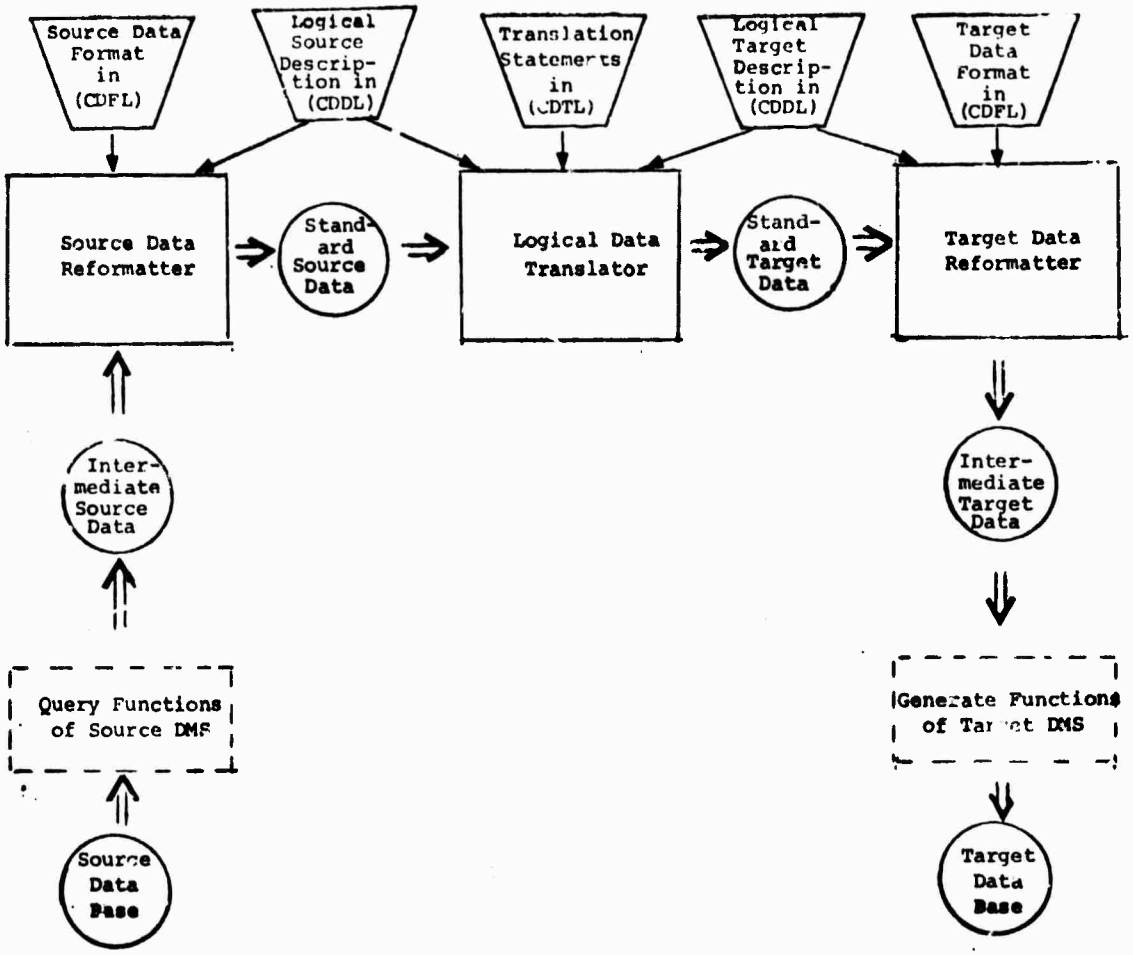


Figure 4-1. The Data Conversion Process

The conversion system uses the following three languages as shown in Figure 4

1. A common data description language (CDDL). This language is used to express only the logical properties of data bases. The user can describe in it how fields are grouped together, the relationship between groups, and field properties.

2. A common data translation language (CDTL). This language expresses logical restructuring functions, primarily in terms of field-to-field mappings. Functions included are repetition and elimination of field values, creation and elimination of group levels, and modification of data values. In addition, the user can describe the concatenation

source fields into one target field, subset the records to be converted, and order the records after conversion. A more detailed description of these functions is given in [7].

3. A common data format language (CDFL). Statements in this language are used by the reformatting processor at both the source and target ends. In this language, the user specifies the input and output format conventions used by the target and source DMSs, respectively.

4.1.3 Present Level of Accomplishment

Most of the work to date has concentrated on the central component of the system: the logical data translator. The translator comprises two main components: the Analyzer and the Restructurer. The Analyzer performs syntax analysis on the CDDL and CDTL statements and semantic analysis to determine whether translation requests are legal. The restructurer uses a conversion table generated by the Analyzer to convert the source records into target records. A prototype of the logical translator is now implemented on SDC's VM-370/145 system.

In addition, source reformatters were built for files in TDMS (an SDC DMS) and for sequential files. Target reformatters were built for ORBIT (an SDC bibliographic search system) and for report display. The reformatters and the translator were used to convert and restructure several large data bases. The system is highly efficient; current tests show that a data base of 5 million bytes is converted in about one minute of CPU time.

4.2 MAJOR ACCOMPLISHMENTS FOR 1974-1975

Major accomplishments during this contract year were made in the design, implementation, and performance testing of the several elements of the conversion system. Because we wanted to demonstrate that our approach leads to a practical, efficient, user-oriented conversion system, the implementation of the system and the demonstration of actual data base conversions were the major tasks for the year. Before expanding on accomplishments during the year, we describe briefly the status of the project at the beginning of the year.

After our approach was selected and specified, the Common Data Description Language (CDDL) and the Common Data Translation Language (CDTL) were defined. Defining the CDDL was an easy task, since it involved only a representation of logical structures of data. Defining the CDTL was a major task that included the selection of the desired restructuring functions and their representation in a user-oriented form. In addition, a set of semantic rules were developed to ensure that a combination of restructuring functions specified by a user produces a semantically meaningful target data base description. These accomplishments were described in our final report to ARPA for 1973-74 [10]. Another design that was already accomplished was that of the "standard form." Also, modules that can read and write a stream of data in the standard form were developed. A description of the "standard form" and the considerations affecting its design is given in [11].

With this at our disposal, we proceeded with the tasks of designing and implementing the several components of the conversion system. The following sections describe the operation of these components as presently implemented.

4.2.1 The Analyzer

The Analyzer is responsible for a syntax parsing and analysis of the CDDL and CDTL statements and for performing a semantic analysis of the restructuring functions according to the set of semantic rules. (A description of the rules is given in [7].) An algorithm was developed according to these rules and was incorporated in the Analyzer. The output of the Analyzer is a conversion table, which is used by the Restructurer for actual restructuring of the data stream.

The Analyzer is diagrammed in Figure 4-2. It operates as follows. First, syntax analysis is performed on the source and target data description statements (in CDDL). If no errors are found, source and target tables are produced that contain precise information about the data base. If an error is found, an error message is issued to the user. Errors discovered at this stage are more than strictly syntactical; for example, a missing description of a field will be detected at this stage. The next step is the association process, in which source and target fields are associated according to the conversion statements. In this step, the translation statements are also checked for syntax legality. This process produces the association matrix, which is used by the semantic analyzer. The association matrix has an entry for every pair of source and target groups. When a mapping is requested from a field in a source group to a field in a target group, the type of mapping is recorded in the appropriate entry.

The purpose of the semantic analyzer is to determine, from the collection of conversion functions requested by a user, whether the request is semantically meaningful. To determine this, the semantic analyzer examines the association matrix for possible conflicts; if none are found, it determines the correspondence between source and target groups.

After the semantic analysis is found to be correct and the correspondences between source and target groups have been determined, the conversion table is constructed. Every entry in the conversion table contains a coded instruction to the Restructurer to perform one of the translation functions required. The entry includes information about the source field from which a value (or values) is to be extracted, the target field to be created, the conversion function required, and additional operations (such as 'string modification' and 'subset'), if specified. The conversion table is the sole input to the Restructurer.

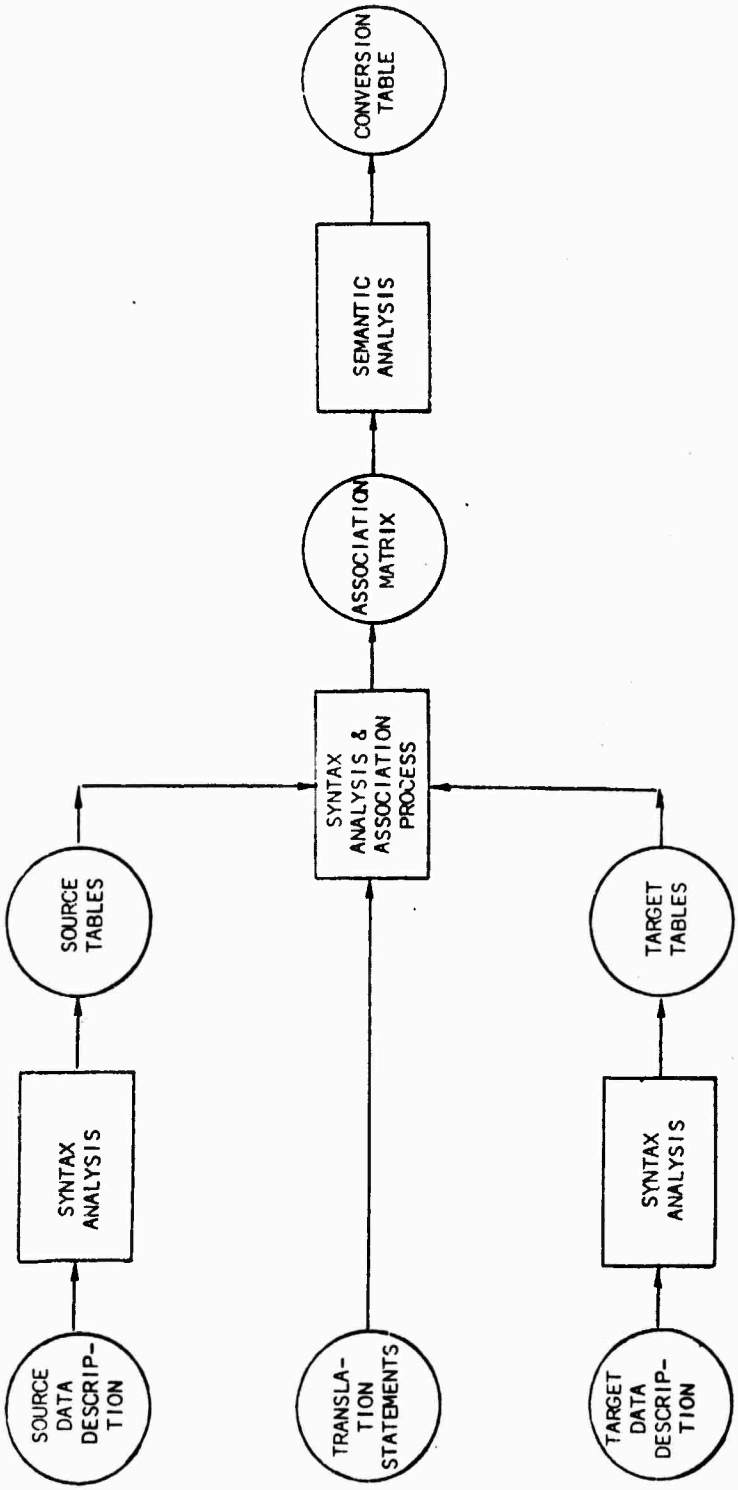


Figure 4-2. The Analyzer

4.2.2 The Restructurer

The Restructurer is diagrammed in Figure 4-3. Basically, it is driven by the conversion table (CTAB) and keeps track of the current CTAB entry. As it proceeds, it also keeps pointers to the current instances of both the source and target data for all the levels of the hierarchy involved.

The controller reads the current CTAB entry to determine which module to call. The different modules correspond to restructuring commands in CDTL (such as DIRECT, REPEAT, GROUP, etc.). These modules in turn call the READER module (possibly more than once) to extract the desired value(s) from the appropriate level of the source hierarchy. The READER uses the pointers embedded in the source standard form to extract data values efficiently. The CONCATENATE module can call on other modules to extract the values to be concatenated. Then the value returned to the controller is written into the target record in the standard form by the WRITER module. The GROUP and END modules are responsible for repositioning the current CTAB entry and the current pointers to the source and target data when a new (lower-level) target group is to be formed or the current group is to be "closed." Some of the modules mentioned above can call additional modules to perform lower-level functions, such as string modification or subset. All modules except the LEVELUP module have now been implemented.

The controller continues to move up and down the CTAB entries until all source instances have been exhausted. Then it gets the next source record and repeats the operation. When all source records have been processed, the restructuring process terminates. Since the Restructurer produces the target data in the standard form, these data can be used again as input for an additional pass of restructuring if necessary. Multiple-pass restructuring is sometimes useful for complex conversions that cannot be readily expressed in CDTL because of our desire to keep that language simple enough to be used by applications programmers.

4.2.3 The Reformatter

The reformatters do not perform any restructuring of data. Rather, they perform a one-to-one mapping of values and instances to and from the standard form. On the input side, the source reformatter is responsible for locating the source values and instances, using the source data description statements, and generating the equivalent standard form. On the output side, the target reformatter is responsible for generating records in a format acceptable to the target system, using the standard form and the target data description statements. In either case, a description of the format (input or output) is necessary.

Rather than have a single reformatter for all types of formats, we found it preferable to classify the reformatters by type of formats. There are two major categories: the pair type and the report type. In the pair type, the

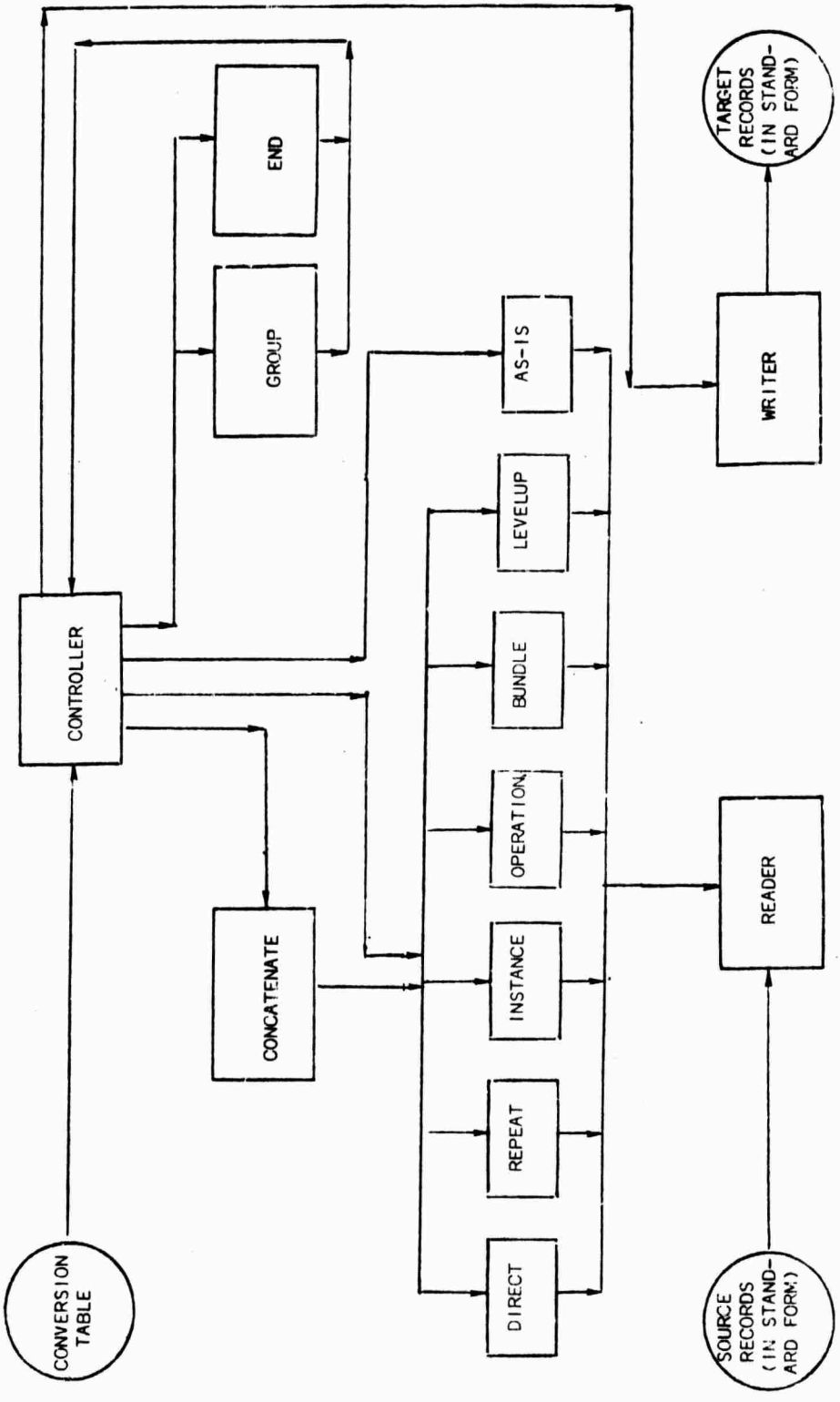


Figure 4-3. The Restructurer

data base is represented as a contiguous data stream, with values being preceded by field identifiers. The field-value pair identifies uniquely the field that the value belongs to. In addition, group identifiers designate new instances, and group or record terminators are also sometimes used. A group identifier can be the field or group name, a number, or another designator assigned to the group. The report type format can typically be found in the output from data management systems. In this type, fields and groups are assigned positions (such as column number), and the start and end of instances follow some convention (e.g., two line-feeds). A variation of this type, although not common, is the use of separator markers to separate values or instances.

Another important format type that should be considered is the sequential type. It is typically found in COBOL or PL/1 sequential files and requires a language to describe its characteristics. It is important to have a reformatter for this type when we wish to handle data bases that were not generated by a data management system. A language for the sequential format was investigated by Housel, Smith, Shu, and Lum [12], who have taken a similar approach to data conversion [13]. The sequential format must also deal with physical characteristics of the data that depend on the particular computer hardware involved (such as the physical representation of numbers).

In order to test the converter and experiment with large data bases, we have built several reformatters. We found it practical to have different reformatters for the different format types.

For the input, we implemented two types of reformatters. The first was a pair type that could be used with files generated by TDMS (an SDC DMS). The other was a generalized reformatter for directory-type sequential files.* These are sequential files organized with a directory in front of the different records types. Each directory has a predefined number of blocks, and each block contains information about the value of a given field. Since this is too much detailed information to be specified by means of parameters, a language definition was necessary. Essentially, the language consists of global statements for the directory and the blocks and local statements associated with each group and field of the data base.

For the output, we implemented a reformatter to a bibliographic search system called ORBIT. (ORBIT employs a format so different from those described above that a special formatter had to be built specifically for it.) The other reformatter generated a report from the "standard form" and is used to display records at each stage of the conversion process. This was a useful tool for debugging, and it is also useful for displaying a subset of the records to be converted before committing an entire data base for conversion.

*By "generalized" we mean a reformatter that is driven by a language or by a set of parameters to describe the format of the data stream. This is in contrast to a special-purpose reformatter for each and every data stream.

4.2.4 Experimentation and Performance

In addition to multiple conversions of small experimental data bases, two large data bases were converted. The small data bases were used in conjunction with numerous combinations of conversion functions in order to test the restructurer fully. The large data base conversions were done primarily for performance measurements. Two large existing data bases (several million bytes each) were converted. One was a TDMS data base with very large records (our system can accommodate records up to 70K bytes); the other was a directory-type sequential file with bibliographic information. The results demonstrated the 5-million-bytes-per-CPU-minute conversion rate mentioned earlier.

4.3 CONCLUSIONS AND RECOMMENDATIONS

The logical-level approach developed by this project proved to be very successful in that it provides practical, useful, and efficient user-oriented tools for data base conversion and restructuring. With a relatively small effort, we have shown that efficient tools can be implemented and used for converting even very large data bases.

The dissociation of the conversion process from the storage and physical representations of data led to the definition of languages (CDDL, CDTL) that are simple enough to be used effectively by a programmer who is not sophisticated in dealing with the internal structures of computing systems. No knowledge is required of inversion tables or of the hashing of data elements; all that is required is a knowledge of the logical organization of the data base to be converted and of the format of the data stream. The restructuring functions were designed to be intuitive, involving primarily field-to-field mappings. Practical considerations, such as the ability to convert only a selected number of records (rather than committing an entire data base for conversion) and the ability to run any of the system components separately or together, were also implemented for user convenience. These facilities are described in a user's guide [14].

We recommend further work in two areas:

1. The development of generalized reformatters for both input and output for the different format types. We concluded that multiple reformatters for the different types of formats will be more practical and less confusing to the user. The reason is that different format types have very little in common.

2. The development of mechanisms for multiple-hierarchy correlation. This is necessary in order to accommodate network and relational data bases. Although most existing data management systems deal only with hierarchical data bases, the trend is towards more generalized data structures.

4.4 STAFF

Dr. Arie Shoshani, Project Leader
Kenneth M. Brandon

4.5 PUBLICATIONS AND REFERENCES

[1] Merten, A. G., and J. P. Fry, "A Data Description Language Approach to File Translation," 1974 ACM SIGFIDET Workshop, pp. 191-205.

[2] Ramirez, J. A., N. A. Rin, and N. S. Prywes, "Automatic Generation of Data Conversion Programs Using a Data Description Language," 1974 ACM SIGMOD Workshop, San Jose, pp. 207-227.

[3] Sibley, E. H., and R. W. Taylor, "A Data Definition and Mapping Language," CACM 16(12), December, 1973, pp. 750-759.

[4] Fry, J. P., R. L. Frank, and E. A. Hershey III, "A Developmental Model for Data Translation," 1972 ACM SIGFIDET Workshop, pp. 77-106.

[5] Smith, D. P., "A Method for Data Translation Using the Stored Data Definition and Translation Task Group Languages," 1972 ACM SIGFIDET Workshop, pp. 67-77.

[6] Smith, D. P., "An Approach to Data Description and Conversion," Ph.D., Dissertation, University of Pennsylvania, 1971.

[7] Shoshani, A., "The Logical Approach to Data Base Conversion," 1975 ACM SIGMOD Workshop on Management of Data Control, San Jose, May, 1975, pp. 112-122.

[8] Shoshani, A., and K. Brandon, "On the Implementation of a Logical Data Base Converter," presented at the International Conference on Very Large Data Bases, Boston, September, 1975.

[9] Fogt, K., Data Structure Levels in Information Systems, System Development Corporation Report No. TM-5123, June, 1973.

[10] Bernsteir, M. I., Final Report to the Director, Advanced Research Projects Agency, for the Period 1 October 1973 to 15 September 1974, System Development Corporation Report No. TM-5243/002/00, November 15, 1974.

[11] Shoshani, A., and K. Brandon, The Implementation of a Logical Data Base Converter, System Development Corporation Report No. TM-5590, October, 1975.

[12] Housel, B. C., D. P. Smith, N. C. Shu, and V. Y. Lum, "DEFINE: A Nonprocedural Data Description Language for Defining Information Easily," Proceedings of ACM Pacific '75 Symposium, April, 1975, pp. 62-70.

[13] Shu, N. C., B. C. Housel, and V. Y. Lum, "CONVERT: A High Level Translation Definition Language for Data Conversion," presented at the 1975 ACM SIGMOD Workshop on the Management of Data, San Jose, May, 1975 (published in CACM 18(10) October, 1975, pp. 557-567).

[14] Brandon, K. M., CODS User Guide, System Development Corporation Report No. TM-5600, November, 1975.